УКРАЇНА

ЖИТОМИРСЬКИЙ КООПЕРАТИВНИЙ ФАХОВИЙ КОЛЕДЖ БІЗНЕСУ І ПРАВА

НАВЧАЛЬНА ПРАКТИКА З ОСНОВ ПРОГРАМУВАННЯ

Методичні рекомендації для самостійної роботи студентів

Спеціальність: F2 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

Укладач: Олександр ФІЛІНСЬКИЙ – спеціаліст

Затверджено
на засіданні циклової комісії інженерії
програмного забезпечення та математики
Протокол № ___ від «____» _____ 2026 р.
Голова комісії _____ Наталія ГОНГАЛО

Житомир 2026 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

Методичні рекомендації для самостійної роботи студентів спеціальності F2 «Інженерія програмного забезпечення» складено відповідно до програми обов'язкової навчальної дисципліни «Навчальна практика з основ програмування», розробленої для підготовки фахових молодших бакалаврів за освітньо-професійною програмою «Інженерія програмного забезпечення».

Самостійна робота студента — це форма організації освітнього процесу, під час якої заплановані завдання виконуються студентом під методичним керівництвом викладача, але без його безпосередньої участі. Згідно з навчальним планом, на самостійне опрацювання дисципліни відведено 108 годин (загальний обсяг дисципліни — 180 годин, з них 72 години практичних занять).

Мета методичних рекомендацій — надати студентам допомогу під час опрацювання питань дисципліни, винесених на самостійне вивчення, та сприяти формуванню стійких практичних навичок алгоритмізації та програмування мовою C++ — від найпростіших програм до завершеного програмного застосунку з об'єктно-орієнтованою структурою.

У методичних рекомендаціях за кожною з шістнадцяти тем дисципліни наведено перелік питань для самостійного вивчення, методичні рекомендації до їх опрацювання, практичне завдання для самостійної роботи студента (СРС) та запитання для самоперевірки. Наприкінці подано критерії оцінювання та список рекомендованої літератури й інформаційних ресурсів; номери джерел у розділі «Література» кожної теми відповідають нумерації цього списку.

Практичні завдання виконуються в середовищі розробки Dev-C++ (компілятор TDM-GCC) або в онлайн-компіляторі OnlineGDB (https://www.onlinegdb.com).

Опрацьовувати питання самостійної роботи рекомендується в такій послідовності:

– ознайомитися з темою та питаннями для самостійного вивчення;
– вивчити питання, скориставшись методичними рекомендаціями та рекомендованою літературою;
– скласти стислий конспект опрацьованих питань;
– виконати практичне завдання, перевірити роботу програми на кількох наборах даних і зберегти програмний код;
– дати відповіді на запитання для самоперевірки.

Навчальний матеріал для самостійного вивчення виноситься на поточний і підсумковий контроль знань. За бажанням викладач може замінити або доповнити запитання для самоперевірки завданнями: підготувати реферат, презентацію, виконати тестування тощо.

Розподіл годин самостійної роботи за темами наведено в таблиці 1.

Таблиця 1

Розподіл годин самостійної роботи за темами

№	Назва теми	Кількість годин
1	Вступ до навчальної практики. Середовище розробки мовою С++	4
2	Структура програми мовою С++. Типи даних, змінні та константи. Введення й виведення даних	6
3	Операції та вирази. Перетворення типів. Лінійні алгоритми	4
4	Алгоритми розгалуження. Умовний оператор та оператор вибору	6
5	Циклічні алгоритми. Оператори циклу	12
6	Одновимірні масиви. Алгоритми пошуку та сортування	12
7	Двовимірні масиви. Опрацювання матриць	4
8	Символьні дані та рядки. Опрацювання текстової інформації	6
9	Функції. Модульна структура програми	6
10	Рекурсія. Рекурсивні алгоритми	4
11	Покажчики, посилання та динамічний розподіл пам’яті	6
12	Структури, перелічення та контейнери стандартної бібліотеки	4
13	Робота з файлами. Опрацювання виключних ситуацій. Тестування програм	6
14	Основи об’єктно-орієнтованого програмування. Класи та об’єкти	12
15	Успадкування та поліморфізм	8
16	Практикум з програмування: розроблення та захист прикладного проєкту	8
	Разом	**108**

ТЕМА 1. ВСТУП ДО НАВЧАЛЬНОЇ ПРАКТИКИ. СЕРЕДОВИЩЕ РОЗРОБКИ МОВОЮ C++

Література: *2, 4; інформаційні ресурси: 15, 16, 17, 21, 26.*

Кількість годин для самостійної роботи: *4.*

Питання для самостійного вивчення:

1. Історія створення мови C++ та етапи її розвитку. Стандарти мови (C++11, C++14, C++17, C++20).
2. Компілятор, компонувальник та інтегроване середовище розробки. Огляд компіляторів (GCC/MinGW, MSVC, Clang) і середовищ розробки.
3. Встановлення та налаштування середовища Dev-C++ на домашньому комп'ютері. Робота в онлайн-компіляторі OnlineGDB.
4. Система контролю версій Git: призначення та базові команди (init, add, commit, status, log). Сервіс GitHub.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Розпочніть з історії мови. C++ створив Б'ярн Страуструп на початку 1980-х років як розширення мови C засобами об'єктно-орієнтованого програмування. Відтоді мова розвивається через міжнародні стандарти: C++98, C++11 (найбільше оновлення — `auto`, цикл range-based for, `nullptr`), C++14, C++17, C++20. Зверніть увагу, що різні компілятори підтримують різні стандарти, тому в Dev-C++ доцільно явно ввімкнути стандарт C++11 (меню **Tools → Compiler Options**, параметр `-std=c++11`). Складіть короткий огляд стандартів у вигляді таблиці: рік, головні нововведення.

З'ясуйте різницю між компілятором, компонувальником та середовищем розробки. Компілятор (g++, MSVC, Clang) перевіряє текст програми на помилки та перетворює його на об'єктний код, компонувальник збирає з об'єктних файлів і бібліотек виконуваний файл `.exe`, а інтегроване середовище (Dev-C++, Code::Blocks, Visual Studio, CLion) об'єднує редактор коду, компілятор і відлагоджувач в одному вікні. Порівняйте кілька

середовищ за критеріями: вартість, складність встановлення, вбудований компілятор, наявність відлагоджувача.

Встановіть Dev-C++ на домашній комп'ютер. Під час встановлення оберіть повний набір компонентів (Full), а після першого запуску — українську або англійську мову інтерфейсу. Перевірте роботу: створіть файл через **File → New → Source File**, збережіть його з розширенням `.cpp`, наберіть найпростішу програму та запустіть клавішею F11. Якщо встановлення неможливе, зареєструйтеся в онлайн-компіляторі OnlineGDB — він працює у браузері й не потребує встановлення.

Ознайомтеся із системою контролю версій Git. Git зберігає історію змін програмного коду: кожна збережена версія (коміт) містить опис змін і дає змогу будь-коли повернутися до попереднього стану. Встановіть Git, створіть обліковий запис на GitHub і виконайте в окремій папці базові команди: `git init` (створити репозиторій), `git add .` (додати файли), `git commit -m "Перша програма"` (зберегти версію), `git status` і `git log` (переглянути стан та історію). Навички роботи з Git знадобляться під час виконання індивідуального проєкту.

Повторіть правила охорони праці під час роботи з комп'ютером: правильна відстань до монітора (50–70 см), перерви кожні 45–60 хвилин, вправи для очей, заборона самостійно відкривати системний блок та торкатися пошкоджених кабелів.

ПРАКТИЧНЕ ЗАВДАННЯ

Встановити на домашньому комп'ютері Dev-C++ (або зареєструватися в OnlineGDB) та написати програму, яка виводить інформацію про студента: прізвище та ім'я, групу, спеціальність, назву коледжу та рік вступу — кожен пункт з нового рядка в рамці із символів *. Підготувати в текстовому редакторі порівняльну таблицю «Компілятори та середовища розробки C++» (не менше 4 позицій) за критеріями: розробник, вартість, підтримувані ОС, наявність відлагоджувача. Створити локальний Git-репозиторій, додати до

нього файл програми та виконати перший коміт; результат команди `git log` записати в конспект.

Запитання для самоперевірки:

1. Які основні нововведення з'явилися в стандарті C++11?
2. Чим компілятор відрізняється від компонувальника? Які файли створює кожен із них?
3. Для чого програмісту потрібна система контролю версій Git? Що таке коміт?

ТЕМА 2. СТРУКТУРА ПРОГРАМИ МОВОЮ C++. ТИПИ ДАНИХ, ЗМІННІ ТА КОНСТАНТИ. ВВЕДЕННЯ Й ВИВЕДЕННЯ ДАНИХ

Література: *2, 4; інформаційні ресурси: 9, 10, 19, 22.*

Кількість годин для самостійної роботи: *6.*

Питання для самостійного вивчення:

1. Етапи побудови програми: препроцесування, компіляція, компонування. Призначення проміжних файлів (.i, .o/.obj, .exe).
2. Базові типи даних, їхній розмір і діапазон значень. Операція sizeof та бібліотека <climits>. Переповнення цілочисельних типів.
3. Форматоване виведення даних: маніпулятори setw, setprecision, fixed, left, right.
4. Правила іменування змінних та оформлення програмного коду.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Розберіться, що відбувається з програмою між натисканням клавіші F11 і появою вікна консолі. Спочатку **препроцесор** обробляє директиви `#include` і `#define` та створює проміжний текст (файл `.i`), потім **компілятор** перетворює його на об'єктний код (`.o`), і нарешті **компонувальник** приєднує бібліотеки та створює виконуваний файл `.exe`. Знайдіть у папці з програмою файл `.exe` і переконайтеся, що його можна запустити подвійним клацанням без середовища розробки.

Опрацюйте базові типи даних: цілі (`short`, `int`, `long long`, `unsigned`), дійсні (`float`, `double`), символьний `char` і логічний `bool`. Розмір типу в байтах визначає операція `sizeof`, а найбільші й найменші значення зберігають константи бібліотеки `<climits>`: `INT_MAX`, `INT_MIN`, `LLONG_MAX` тощо. Складіть таблицю типів зі стовпцями «тип», «розмір, байт», «мінімум», «максимум», заповнивши її за допомогою власної програми, а не з підручника.

Особливу увагу приділіть **переповненню**. Якщо до змінної типу `int` зі значенням `INT_MAX` додати 1, результат «перескочить» на від'ємне число. Компілятор про це не попереджає, тому для великих значень (наприклад, факторіала чи суми великої кількості чисел) обирайте тип `long long`. Перевірте це на практиці та запишіть висновок у конспект.

Навчіться керувати виглядом результатів. Маніпулятори бібліотеки `<iomanip>` задають ширину поля (`setw(10)`), кількість знаків після коми (`fixed << setprecision(2)`) та вирівнювання (`left`, `right`). Пам'ятайте: `setw` діє лише на одне наступне значення, а `fixed` і `setprecision` — до скасування. Саме за допомогою цих маніпуляторів будують акуратні таблиці з вирівняними стовпцями.

Засвойте правила оформлення коду: змістовні імена змінних латиницею (`price`, `totalSum`, а не `a1`, `x2`), один оператор у рядку, відступ у 4 пробіли всередині фігурних дужок, коментарі до кожного логічного блоку. Акуратно оформлений код легше читати, перевіряти та виправляти.

ПРАКТИЧНЕ ЗАВДАННЯ

Скласти програму «Таблиця типів даних», яка за допомогою `sizeof` і констант `<climits>` виводить таблицю для типів `char`, `short`, `int`, `long long`, `unsigned int`, `float`, `double` (назва типу, розмір у байтах, мінімальне та максимальне значення) з вирівнюванням стовпців через `setw`. Окремо написати програму-візитівку, яка запитує в користувача ім'я, вік, зріст (дійсне число) та улюблену мову програмування й виводить ці дані у вигляді

відформатованої картки. Дослідити переповнення: вивести значення `INT_MAX` і `INT_MAX + 1` та пояснити результат у коментарі.

Запитання для самоперевірки:

1. Які етапи проходить програма від тексту `.cpp` до виконуваного файлу `.exe`?
2. Що таке переповнення цілочисельного типу і як його уникнути?
3. Чим відрізняється дія маніпулятора `setw` від дії `setprecision`?

ТЕМА 3. ОПЕРАЦІЇ ТА ВИРАЗИ. ПЕРЕТВОРЕННЯ ТИПІВ. ЛІНІЙНІ АЛГОРИТМИ

Література: *2, 4; інформаційні ресурси: 10, 19, 22.*

Кількість годин для самостійної роботи: *4.*

Питання для самостійного вивчення:

1. Пріоритет та асоціативність операцій мови C++.
2. Неявне та явне перетворення типів. Втрата точності під час перетворення. Операція static_cast.
3. Математичні функції бібліотеки <cmath>. Генерування псевдовипадкових чисел (rand, srand).
4. Побудова блок-схем лінійних алгоритмів.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Лінійний алгоритм — це послідовність дій, які виконуються одна за одною без жодних умов і повторень: введення даних → обчислення → виведення результату. Перш ніж писати програму, навчіться зображувати такий алгоритм блок-схемою: овал «Початок», паралелограм «Введення», прямокутник «Обчислення», паралелограм «Виведення», овал «Кінець».

Опрацюйте таблицю пріоритетів операцій. Множення, ділення та остача (`*`, `/`, `%`) виконуються раніше за додавання й віднімання, а присвоєння — останнім. Обчисліть кілька виразів спочатку вручну, а потім перевірте себе програмою, наприклад: `7 + 3 * 2`, `(7 + 3) * 2`, `17 / 5 * 5 + 17 % 5`.

Розбіжності між вашим і комп'ютерним результатом вкажуть на прогалини в розумінні пріоритетів.

Найчастіша помилка початківців — **цілочисельне ділення**. Якщо обидва операнди цілі, результат теж цілий: `7 / 2` дорівнює 3, а не 3.5. Щоб отримати дійсний результат, хоча б один операнд має бути дійсним (`7.0 / 2`) або його слід явно перетворити: `static_cast<double>(a) / b`. Дослідіть і зворотну ситуацію — присвоєння дійсного значення цілій змінній, під час якого дробова частина просто відкидається.

Ознайомтеся з бібліотекою `<cmath>`: `sqrt` (корінь), `pow` (степінь), `fabs` (модуль), `sin`, `cos` (аргумент у радіанах!), `round`, `floor`, `ceil`. Для моделювання випадкових величин використовують `rand()` з бібліотеки `<cstdlib>`: вираз `rand() % 6 + 1` дає випадкове число від 1 до 6. Щоб числа відрізнялися під час кожного запуску, на початку програми один раз викличте `srand(time(0))`.

Потренуйтеся в опрацюванні цифр числа: остання цифра — `n % 10`, число без останньої цифри — `n / 10`. Цей прийом знадобиться в багатьох задачах наступних тем.

ПРАКТИЧНЕ ЗАВДАННЯ

Розробити програму «Конвертер одиниць», яка запитує в користувача кількість секунд, відстань у метрах і температуру за Цельсієм та виводить: час у форматі «години : хвилини : секунди», відстань у кілометрах і милях (2 знаки після коми), температуру за Фаренгейтом і Кельвіном. Окремо скласти програму, що обчислює суму та добуток цифр введеного чотиризначного числа і число, записане цими цифрами у зворотному порядку. Для однієї з програм побудувати блок-схему (від руки або в редакторі draw.io).

Запитання для самоперевірки:

1. Чому вираз `7 / 2` у мові C++ дорівнює 3? Як отримати 3.5?
2. Для чого використовують операцію `static_cast`?
3. Як отримати випадкове ціле число в діапазоні від 10 до 20?

ТЕМА 4. АЛГОРИТМИ РОЗГАЛУЖЕННЯ. УМОВНИЙ ОПЕРАТОР ТА ОПЕРАТОР ВИБОРУ

Література: *2, 4; інформаційні ресурси: 11, 19, 22.*

Кількість годин для самостійної роботи: *6.*

Питання для самостійного вивчення:

1. Логічні операції &&, ||, !. Таблиці істинності. Скорочене обчислення логічних виразів.
2. Складені умови: належність числа проміжку, високосний рік, тип трикутника, чверть координатної площини.
3. Перетворення конструкції if–else if в оператор switch та навпаки.
4. Побудова блок-схем розгалужених алгоритмів у редакторі draw.io.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Розгалужений алгоритм обирає одну з кількох дій залежно від умови. На блок-схемі умову зображують ромбом, з якого виходять дві стрілки — «так» і «ні». Почніть з побудови блок-схеми кожної задачі, і лише потім переходьте до коду: так значно легше не загубити жодної гілки.

Складіть таблиці істинності для операцій `&&` (І), `||` (АБО) та `!` (НЕ). Запам’ятайте: умова з `&&` істинна лише тоді, коли істинні обидві частини, а з `||` — коли істинна хоча б одна. Типова помилка — запис подвійної нерівності як у математиці: `0 < x < 10`. У C++ це працює неправильно; правильний запис: `x > 0 && x < 10`.

Дослідіть **скорочене обчислення**: якщо ліва частина `&&` хибна, права вже не перевіряється. Цим користуються для захисту від помилок, наприклад `if (b != 0 && a / b > 2)` — ділення відбудеться лише тоді, коли дільник не нуль.

Розберіть типові задачі на складені умови. Рік є високосним, якщо він ділиться на 4 і не ділиться на 100, або ділиться на 400: `(y % 4 == 0 && y % 100 != 0) || y % 400 == 0`. Трикутник зі сторонами a, b, c існує, якщо кожна сторона менша за суму двох інших. Для квадратного рівняння

спочатку обчисліть дискримінант, а потім розгляньте три випадки: D > 0, D = 0, D < 0.

Опрацюйте оператор вибору `switch`. Він порівнює цілочисельне або символьне значення з кількома константами `case`. Не забувайте оператор `break` наприкінці кожної гілки — без нього виконання «провалиться» в наступний `case`. Гілка `default` виконується, якщо жоден варіант не підійшов. Потренуйтеся перетворювати ланцюжок `if-else if` на `switch` і навпаки та зробіть висновок, коли кожна конструкція зручніша.

ПРАКТИЧНЕ ЗАВДАННЯ

Розробити програму «Калькулятор», яка запитує два дійсні числа та знак операції (`+`, `-`, `*`, `/`, `%` для цілих частин чисел, `^` — піднесення до степеня) і за допомогою `switch` виводить результат. Передбачити повідомлення про ділення на нуль і невідому операцію. Окремо написати програму, яка за введеними координатами точки (x, y) визначає, в якій чверті координатної площини вона лежить або на якій осі. Для обох програм побудувати блок-схеми в редакторі draw.io та зберегти їх як зображення.

Запитання для самоперевірки:

1. Чому умову «x від 0 до 10» не можна записати як `0 < x < 10`?
2. Що станеться, якщо в операторі `switch` пропустити `break`?
3. Як працює скорочене обчислення логічних виразів і яку користь воно дає?

ТЕМА 5. ЦИКЛІЧНІ АЛГОРИТМИ. ОПЕРАТОРИ ЦИКЛУ

Література: *2, 4; інформаційні ресурси: 7, 11, 19, 21.*

Кількість годин для самостійної роботи: *12.*

Питання для самостійного вивчення:

1. Трасування циклів: ручне виконання з таблицею значень змінних.
2. Обчислення сум і добутків. Наближене обчислення значень функцій рядами із заданою точністю.

3. Перевірка числа на простоту. Пошук НСД (алгоритм Евкліда). Виведення фігур із символів за допомогою вкладених циклів.
4. Типові помилки в циклах (нескінченний цикл, помилка на одиницю). Робота з відлагоджувачем: точки зупину, покроковий режим, спостереження за змінними.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Цикл повторює одні й ті самі дії кілька разів. Оберіть вид циклу залежно від задачі: `for` — коли кількість повторень відома заздалегідь (наприклад, від 1 до n), `while` — коли цикл триває, поки виконується умова, а кількість кроків невідома, `do-while` — коли тіло потрібно виконати хоча б один раз (наприклад, повторне введення до отримання правильного значення).

Обов'язково навчіться **трасувати** цикли вручну. Накресліть таблицю, де стовпці — змінні, а рядки — кроки циклу, і заповнюйте її так, ніби ви комп'ютер. Наприклад, для обчислення суми 1 + 2 + … + 5 у таблиці будуть стовпці `i` і `sum`. Цей навик допомагає зрозуміти, чому програма дає неправильний результат, ще до її запуску.

Опрацюйте накопичення сум і добутків: перед циклом змінну суми ініціалізують нулем, а добутку — одиницею. Для наближеного обчислення рядами (наприклад, $e \approx 1 + 1/1! + 1/2! + \ldots$) цикл `while` триває, доки черговий доданок більший за задану точність ε. Кожен наступний доданок зручно отримувати з попереднього, а не обчислювати факторіал щоразу заново.

Розберіть класичні алгоритми. Число n просте, якщо воно не ділиться на жодне число від 2 до $\sqrt{n}$. Алгоритм Евкліда знаходить НСД так: поки $b \neq 0$, замінюємо пару (a, b) на (b, a % b); коли b стане нулем, відповідь — a. Вкладені цикли використовуйте для виведення фігур: зовнішній цикл відповідає за рядки, внутрішній — за символи в рядку.

Вивчіть типові помилки. **Нескінченний цикл** виникає, коли змінна в умові не змінюється всередині тіла циклу; зупинити таку програму можна комбінацією Ctrl + C. **Помилка на одиницю** — коли цикл виконується на

один раз більше або менше, ніж потрібно (`i < n` замість `i <= n`). Для пошуку таких помилок опануйте відлагоджувач Dev-C++: увімкніть у налаштуваннях компілятора генерацію налагоджувальної інформації, поставте точку зупину клацанням по номеру рядка, запустіть режим **Debug** (F5) і виконуйте програму по кроках кнопкою **Next line**, додавши змінні до вікна спостереження (**Add watch**).

ПРАКТИЧНЕ ЗАВДАННЯ

Скласти три програми. 1) Табулювання функції $y = x^2 - 3x + 2$ на відрізку [a; b] з кроком h, які вводить користувач; результат вивести у вигляді таблиці з двома стовпцями та знайти найбільше й найменше значення y. 2) Програма, що виводить усі прості числа в діапазоні від 2 до n та їхню кількість, а також НСД двох введених чисел за алгоритмом Евкліда. 3) Виведення «ялинки» із символів * висотою n рядків за допомогою вкладених циклів. Для програми 2 заповнити в конспекті таблицю трасування алгоритму Евкліда для чисел 48 і 18 та перевірити її за допомогою відлагоджувача.

Запитання для самоперевірки:

1. У яких випадках доцільно використовувати цикл `for`, а в яких — `while` чи `do-while`?
2. Що таке нескінченний цикл і помилка на одиницю? Як їх виявити?
3. Як за допомогою відлагоджувача Dev-C++ поставити точку зупину та переглянути значення змінної?

ТЕМА 6. ОДНОВИМІРНІ МАСИВИ. АЛГОРИТМИ ПОШУКУ ТА СОРТУВАННЯ

Література: *2, 4; інформаційні ресурси: 11, 14, 19, 20.*

Кількість годин для самостійної роботи: *12.*

Питання для самостійного вивчення:

1. Типові алгоритми опрацювання масивів: заповнення випадковими числами, статистика, пошук екстремумів, реверс, циклічний зсув, вставлення та вилучення елементів.
2. Лінійний та двійковий пошук. Підрахунок кількості порівнянь.
3. Алгоритми сортування: бульбашкове, вибором, вставленням. Експериментальне порівняння їхньої ефективності.
4. Вихід за межі масиву та його наслідки.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Масив — це набір однотипних елементів, що зберігаються в пам’яті поспіль і мають спільне ім’я. Доступ до елемента здійснюється за індексом, який у C++ починається з **нуля**: у масиві `int a[10]` є елементи від `a[0]` до `a[9]`. Розмір статичного масиву задають константою, тому зручно оголошувати `const int N = 10;` і далі використовувати `N` у всіх циклах.

Опрацюйте типові алгоритми. Для пошуку максимуму спочатку вважають найбільшим перший елемент, а потім порівнюють з ним решту. Реверс виконують обміном симетричних елементів `a[i]` і `a[n - 1 - i]` до середини масиву. Під час вилучення елемента всі наступні зсувають на одну позицію ліворуч, під час вставлення — праворуч, починаючи з кінця. Обов’язково перевіряйте кожен алгоритм на масиві з 5–6 елементів, виписуючи стан масиву після кожного кроку.

Порівняйте алгоритми пошуку. **Лінійний пошук** послідовно переглядає всі елементи і працює з будь-яким масивом. **Двійковий пошук** працює лише з **відсортованим** масивом: на кожному кроці порівнює шукане значення із середнім елементом і відкидає половину масиву. Для масиву з 1000 елементів лінійний пошук може виконати до 1000 порівнянь, двійковий

— не більше 10. Додайте до своїх програм лічильник порівнянь і переконайтеся в цьому експериментально.

Розберіть три алгоритми сортування. **Бульбашкове**: сусідні елементи порівнюються та обмінюються, і найбільший «спливає» в кінець. **Вибором**: на кожному кроці знаходять мінімум серед невідсортованої частини і ставлять його на місце. **Вставленням**: кожен наступний елемент вставляють у правильну позицію серед уже впорядкованих. Виміряйте кількість порівнянь, обмінів і час роботи (функція `clock()` з `<ctime>`) на масивах з 1 000, 5 000 і 10 000 випадкових чисел і занесіть результати до порівняльної таблиці.

Пам’ятайте про **вихід за межі масиву**: компілятор не перевіряє індекси, тому звертання до `a[10]` у масиві з 10 елементів не дасть помилки компіляції, але може зіпсувати інші дані або аварійно завершити програму. Найчастіша причина — умова `i <= N` замість `i < N`.

ПРАКТИЧНЕ ЗАВДАННЯ

Розробити програму, яка заповнює масив із 20 елементів випадковими числами від −50 до 50 і виводить: масив; суму, середнє арифметичне, кількість додатних і від’ємних елементів; максимальний і мінімальний елементи та їхні індекси; масив після реверсу та після циклічного зсуву на k позицій праворуч. Окремо реалізувати три алгоритми сортування з підрахунком кількості порівнянь і обмінів та виміром часу; провести експеримент на масивах трьох різних розмірів і оформити в конспекті порівняльну таблицю з висновком, який алгоритм найшвидший.

Запитання для самоперевірки:

1. З якого індексу починається нумерація елементів масиву в C++? Який індекс останнього елемента масиву з N елементів?
2. Чому двійковий пошук не можна застосовувати до невідсортованого масиву?

3. Який з вивчених алгоритмів сортування виявився найефективнішим у вашому експерименті і чому?

ТЕМА 7. ДВОВИМІРНІ МАСИВИ. ОПРАЦЮВАННЯ МАТРИЦЬ

Література: *2, 4; інформаційні ресурси: 11, 14, 19.*

Кількість годин для самостійної роботи: *4.*

Питання для самостійного вивчення:

1. Основи матричної алгебри: додавання, множення, транспонування матриць.
2. Обхід матриці за рядками, стовпцями та діагоналями.
3. Прикладні задачі з табличними даними (відомість оцінок, таблиця продажів, температурні спостереження).
4. Нестандартні способи заповнення матриці: «змійкою», по спіралі, магічний квадрат.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Двовимірний масив (матриця) — це таблиця з рядками та стовпцями: `int a[3][4]` містить 3 рядки по 4 елементи. Елемент `a[i][j]` знаходиться в рядку `i` і стовпці `j`. Для опрацювання матриці використовують два вкладені цикли: зовнішній — за рядками, внутрішній — за стовпцями. Якщо поміняти цикли місцями, обхід відбуватиметься за стовпцями.

Опрацюйте діагоналі квадратної матриці розміром N × N. Елементи головної діагоналі мають однакові індекси (`i == j`), допоміжної — задовольняють умову `i + j == N - 1`. Елементи вище головної діагоналі — `j > i`, нижче — `j < i`. Накресліть матрицю 4 × 4, підпишіть індекси кожної клітинки та переконайтеся в цих правилах.

Повторіть матричні операції. Транспонування — заміна рядків стовпцями: `b[j][i] = a[i][j]`. Додавання можливе лише для матриць однакового розміру. Множення матриці A (m × n) на B (n × k) можливе лише тоді, коли кількість стовпців A дорівнює кількості рядків B; кожен елемент результату — сума добутків рядка A на стовпець B, тому множення вимагає

трьох вкладених циклів. Перед виконанням операцій програма має перевіряти сумісність розмірів.

Двовимірні масиви зручні для прикладних таблиць: рядки — студенти, стовпці — предмети; рядки — місяці, стовпці — товари. Для таких задач обчислюйте суми й середні за рядками (окремий одновимірний масив результатів) і за стовпцями. Виводьте матрицю за допомогою `setw`, щоб стовпці були рівними.

ПРАКТИЧНЕ ЗАВДАННЯ

Розробити програму «Відомість успішності»: матриця оцінок 5 студентів з 4 предметів (оцінки від 1 до 12 вводяться з клавіатури або генеруються випадково). Програма виводить таблицю з вирівняними стовпцями, середній бал кожного студента та кожного предмета, найкращого студента і предмет з найнижчим середнім балом. Окремо реалізувати заповнення квадратної матриці N × N числами від 1 до N^2 «змійкою» (парні рядки — зліва направо, непарні — справа наліво) та її транспонування.

Запитання для самоперевірки:

1. За якою умовою визначаються елементи головної та допоміжної діагоналей?
2. За якої умови можна перемножити дві матриці?
3. Як обчислити середнє значення кожного стовпця матриці?

ТЕМА 8. СИМВОЛЬНІ ДАНІ ТА РЯДКИ. ОПРАЦЮВАННЯ ТЕКСТОВОЇ ІНФОРМАЦІЇ

Література: *2, 4; інформаційні ресурси: 9, 10, 18, 19.*

Кількість годин для самостійної роботи: *6.*

Питання для самостійного вивчення:

1. Таблиця ASCII та особливості кодування кирилиці (Windows-1251, UTF-8).
2. Функції бібліотеки <cctype>. Власні реалізації функцій бібліотеки <cstring>.

3. Підрахунок слів і речень, пошук і заміна підрядка, перевірка на паліндром, розбиття рядка на слова, шифр Цезаря.
4. Порівняння рядків у стилі C та класу std::string за зручністю й безпекою.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Тип `char` зберігає не саму літеру, а її **код** — ціле число. Відкрийте таблицю ASCII і знайдіть закономірності: цифри `'0'`–`'9'` мають коди 48–57, великі латинські літери — 65–90, малі — 97–122; різниця між малою і великою літерою становить 32. Тому символ-цифру перетворюють на число виразом `c - '0'`. Українські літери в таблицю ASCII не входять: у Dev-C++ вони кодуються за Windows-1251, в OnlineGDB — за UTF-8 (де одна літера займає 2 байти). Через це функції `<cctype>` правильно працюють лише з латиницею.

Опрацюйте функції `<cctype>`: `isalpha` (літера), `isdigit` (цифра), `isspace` (пробільний символ), `isupper`/`islower` (регістр), `toupper`/`tolower` (зміна регістру). Пам'ятайте, що `toupper` повертає число, тому для виведення символу потрібне перетворення `char(toupper(c))`.

Розберіться з рядками у стилі C: це масив символів, що закінчується нуль-термінатором `'\0'`. Щоб краще зрозуміти, як вони влаштовані, напишіть власні версії функцій `strlen` (довжина — цикл до `'\0'`), `strcpy` (посимвольне копіювання разом із термінатором) і `strcmp` (порівняння до першого розбіжного символу). Порівняйте результати своїх функцій з бібліотечними.

Клас `std::string` значно зручніший: рядок сам зберігає довжину, змінює розмір, порівнюється операцією `==`, об'єднується операцією `+`. Вивчіть методи `length`, `substr(поз, кільк)`, `find` (повертає `string::npos`, якщо нічого не знайдено), `replace`, `insert`, `erase`. Для читання рядка з пробілами використовуйте `getline(cin, s)`, а якщо перед ним було `cin >>`, спочатку викличте `cin.ignore()`.

Опрацюйте прикладні алгоритми: підрахунок слів (слово починається там, де після пробілу йде не пробіл), паліндром (порівняння очищеного рядка з перевернутим), розбиття на слова (`stringstream` або ручний перебір), шифр Цезаря (зсув номера літери в алфавіті за модулем 26: `(c - 'a' + k) % 26 + 'a'`).

ПРАКТИЧНЕ ЗАВДАННЯ

Розробити програму «Текстовий аналізатор», яка зчитує рядок латиницею (з пробілами) і виводить: кількість літер, цифр, пробілів і розділових знаків; кількість слів і найдовше слово; текст із кожним словом, що починається з великої літери; результат перевірки, чи є рядок паліндромом (без урахування пробілів і регістру). Окремо реалізувати шифр Цезаря з ключем k: зашифрувати введений текст, вивести результат і розшифрувати його назад. Реалізувати власну функцію `myStrlen` для рядка у стилі C і порівняти її результат з `strlen`.

Запитання для самоперевірки:

1. Як перетворити символ-цифру `'7'` на число 7 і чому це працює?
2. Чим рядок у стилі C відрізняється від об’єкта класу `string`?
3. Чому функції `<cctype>` неправильно працюють з українськими літерами?

ТЕМА 9. ФУНКЦІЇ. МОДУЛЬНА СТРУКТУРА ПРОГРАМИ

Література: *2, 4; інформаційні ресурси: 5, 10, 19.*

Кількість годин для самостійної роботи: *6.*

Питання для самостійного вивчення:

1. Декомпозиція задачі на функції. Переробка раніше написаних програм із виділенням логіки у функції.
2. Способи передавання параметрів: за значенням, за посиланням, через покажчик.
3. Область видимості змінних. Статичні локальні змінні. Перевантаження функцій і параметри за замовчуванням.

4. Багатофайловий проєкт: заголовкові файли, include guard, аналіз помилок компонування.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Функція — це іменований фрагмент програми, який розв'язує одну конкретну підзадачу. Добра функція має зрозуміле ім'я (`calcAverage`, `printArray`), виконує одну дію і не залежить від глобальних змінних. Візьміть одну з програм попередніх тем (наприклад, обробку масиву) і перепишіть її так, щоб `main()` містила лише виклики функцій: `fillArray`, `printArray`, `findMax`, `sortArray`.

Опрацюйте способи передавання параметрів. **За значенням** (`void f(int x)`) функція отримує копію, і зміни всередині неї не впливають на змінну в `main()`. **За посиланням** (`void f(int &x)`) функція працює з оригіналом. **Через покажчик** (`void f(int *p)`) функція отримує адресу змінної і змінює її через `*p`. Напишіть три версії функції обміну значень двох змінних і поясніть, чому працює лише версія за значенням — ні.

Розберіться з областю видимості. Локальна змінна існує лише всередині свого блоку `{ }`, глобальна — у всій програмі. Статична локальна змінна (`static int count = 0;`) створюється один раз і зберігає значення між викликами функції — так зручно рахувати кількість викликів. Ознайомтеся з **перевантаженням** (кілька функцій з однаковим ім'ям, але різними параметрами) та **параметрами за замовчуванням** (`void line(int n = 20, char c = '-')`), які записують лише в оголошенні функції.

Для великих програм функції розміщують в окремих файлах. Оголошення (прототипи) записують у заголовковий файл `.h`, захищений від повторного включення конструкцією `#ifndef … #define … #endif` або `#pragma once`, а реалізації — у файл `.cpp`. У Dev-C++ для цього створюють проєкт (**File → New → Project → Console Application**) і додають файли через меню **Project**. Навмисно видаліть один файл із проєкту та прочитайте повідомлення компонувальника `undefined reference` — так ви навчитеся розпізнавати помилки компонування.

ПРАКТИЧНЕ ЗАВДАННЯ

Створити багатофайловий проєкт «Бібліотека функцій»: файли `mathutils.h` і `mathutils.cpp` з функціями `isPrime(int)`, `gcd(int, int)`, `factorial(int)` (результат `long long`) та перевантаженою функцією `maxOf` для двох і трьох чисел; файли `arrutils.h` і `arrutils.cpp` з функціями заповнення, виведення та пошуку максимуму в масиві (масив передається разом із розміром). У `main.cpp` продемонструвати роботу всіх функцій. Додати функцію зі статичною локальною змінною, яка виводить номер свого виклику. Пояснити в коментарях, які параметри передаються за значенням, а які — за посиланням.

Запитання для самоперевірки:

1. Чим відрізняється передавання параметра за значенням від передавання за посиланням?
2. Що таке статична локальна змінна і для чого її використовують?
3. Для чого потрібен include guard у заголовковому файлі?

ТЕМА 10. РЕКУРСІЯ. РЕКУРСИВНІ АЛГОРИТМИ

Література: *2, 4; інформаційні ресурси: 14, 19, 20.*

Кількість годин для самостійної роботи: *4.*

Питання для самостійного вивчення:

1. Базовий випадок і крок рекурсії. Стек викликів. Трасування рекурсивних викликів.
2. Рекурсивні алгоритми: сума цифр числа, степінь, НСД, реверс рядка, паліндром, двійковий пошук, ханойські башти.
3. Порівняння рекурсивної та ітеративної реалізацій за швидкодією та витратами пам’яті.
4. Переповнення стеку за відсутності базового випадку.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Рекурсивна функція — це функція, яка викликає сама себе. Кожна правильна рекурсія має дві частини: **базовий випадок** (умова, за якої функція повертає результат без нового виклику) і **крок рекурсії** (виклик для «меншої» задачі, що наближає до базового випадку). Наприклад, для факторіала: базовий випадок `0! = 1`, крок — `n! = n · (n − 1)!`.

Обов'язково трасуйте рекурсію на папері. Намалюйте стек викликів для `fact(4)`: виклики «спускаються» до `fact(0)`, а потім результати «піднімаються» назад: 1 → 1 → 2 → 6 → 24. Для чисел Фібоначчі намалюйте дерево викликів `fib(5)` і порахуйте, скільки разів обчислюється `fib(2)` — ви побачите, чому наївна рекурсія тут дуже повільна.

Потренуйтеся записувати рекурсивно знайомі алгоритми: сума цифр (`n % 10 + sum(n / 10)`), степінь, НСД (`gcd(b, a % b)`), реверс рядка, перевірка паліндрома (порівняння крайніх символів і рекурсія для середини). Розв'яжіть задачу про ханойські башти: щоб перенести n дисків, потрібно перенести n − 1 дисків на допоміжний стрижень, перенести найбільший диск і знову перенести n − 1 дисків. Порахуйте кількість переміщень і переконайтеся, що вона дорівнює $2^n - 1$.

Порівняйте рекурсію та ітерацію. Рекурсивний код часто коротший і ближчий до математичного означення, але кожен виклик займає місце в стеку і потребує часу. Виміряйте час обчислення `fib(35)` рекурсивно та циклом і зробіть висновок. Дослідіть, що станеться, якщо видалити базовий випадок: програма аварійно завершиться через **переповнення стеку** (stack overflow).

ПРАКТИЧНЕ ЗАВДАННЯ

Скласти програму з рекурсивними функціями: сума цифр числа, піднесення до степеня, НСД, реверс рядка, перевірка рядка на паліндром. Для кожної функції написати ітеративний аналог і вивести результати обох версій для 3–4 тестових значень. Реалізувати задачу про ханойські башти з

виведенням усіх переміщень для n = 3 і підрахунком їхньої кількості для n = 10, 15, 20. Виміряти час обчислення числа Фібоначчі з номером 35 рекурсивно та ітеративно і записати висновок у конспект.

Запитання для самоперевірки:

1. З яких двох обов'язкових частин складається рекурсивна функція?
2. Чому рекурсивне обчислення чисел Фібоначчі працює значно повільніше за ітеративне?
3. Що станеться, якщо в рекурсивній функції відсутній базовий випадок?

ТЕМА 11. ПОКАЖЧИКИ, ПОСИЛАННЯ ТА ДИНАМІЧНИЙ РОЗПОДІЛ ПАМ'ЯТІ

Література: *2, 4; інформаційні ресурси: 5, 10, 19.*

Кількість годин для самостійної роботи: *6.*

Питання для самостійного вивчення:

1. Схеми розміщення змінних і покажчиків у пам'яті. Адреси змінних і елементів масиву. Арифметика покажчиків.
2. Обмін значень через покажчик і через посилання. Обробка масивів через покажчики.
3. Динамічні одновимірні та двовимірні масиви. Витік пам'яті та способи його запобігання.
4. Типові помилки: висячий покажчик, подвійне звільнення пам'яті, розіменування nullptr. Покажчики на функції.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Кожна змінна займає ділянку оперативної пам'яті з певною **адресою**. Операція `&x` повертає адресу змінної, а покажчик (`int *p = &x;`) — це змінна, що зберігає таку адресу. Через покажчик можна прочитати або змінити значення: `*p = 10;` (розіменування). Малюйте схеми: прямокутник для змінної, всередині — значення, під ним — адреса; стрілка від покажчика до змінної. Такі схеми найкраще допомагають зрозуміти тему.

Ім’я масиву є покажчиком на його перший елемент, тому `a[i]` і `*(a + i)` — одне й те саме. Під час додавання до покажчика числа 1 адреса збільшується на розмір типу (для `int` — на 4 байти). Виведіть адреси всіх елементів масиву і переконайтеся в цьому. **Посилання** (`int &r = x;`) — це друге ім’я змінної; на відміну від покажчика, воно не може бути порожнім і не змінює об’єкт, на який посилається.

Динамічна пам’ять дає змогу створювати масиви, розмір яких стає відомим лише під час виконання: `int *a = new int[n];`. Після використання пам’ять **обов’язково** звільняють: `delete[] a;`. Динамічну матрицю створюють як масив покажчиків на рядки: спочатку `new int*[rows]`, потім для кожного рядка `new int[cols]`; звільняють у зворотному порядку. Якщо забути `delete`, виникає **витік пам’яті**.

Вивчіть типові помилки та способи їх уникнення: розіменування неініціалізованого покажчика або `nullptr` (аварійне завершення), **висячий покажчик** (звертання до вже звільненої пам’яті — після `delete` присвоюйте покажчику `nullptr`), **подвійне звільнення** однієї ділянки пам’яті, використання `delete` замість `delete[]` для масиву. Як ознайомлювальне питання розгляньте покажчики на функції — вони дозволяють передати функцію як параметр іншій функції.

ПРАКТИЧНЕ ЗАВДАННЯ

Розробити програму, яка запитує в користувача розміри матриці (кількість рядків і стовпців), створює динамічну матрицю, заповнює її випадковими числами, виводить, знаходить суму кожного рядка (результати — у динамічному одновимірному масиві) і коректно звільняє всю виділену пам’ять. Окремо написати функції обміну значень через покажчик і через посилання, функцію обходу масиву за допомогою арифметики покажчиків (без операції `[]`) та програму, що виводить адреси елементів масиву. У конспекті намалювати схему пам’яті для динамічної матриці 3 × 4.

Запитання для самоперевірки:

1. Що означають операції `&x` і `*p`?
2. Чим посилання відрізняється від покажчика?
3. Що таке витік пам’яті і як його уникнути під час роботи з динамічними масивами?

ТЕМА 12. СТРУКТУРИ, ПЕРЕЛІЧЕННЯ ТА КОНТЕЙНЕРИ СТАНДАРТНОЇ БІБЛІОТЕКИ

Література: *1, 2, 4; інформаційні ресурси: 10, 12, 19.*

Кількість годин для самостійної роботи: *4.*

Питання для самостійного вивчення:

1. Структури для предметних областей. Масиви записів: введення, виведення, пошук, фільтрація, сортування за полем.
2. Перелічувані типи enum та enum class.
3. Контейнери std::vector, std::array, std::map, std::set.
4. Алгоритми бібліотеки <algorithm>: sort, find, count, max_element, reverse; функція accumulate.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Структура об’єднує кілька різнотипних даних про один об’єкт: наприклад, студент має ім’я (`string`), курс (`int`) і середній бал (`double`). Оголосіть структуру `struct Student { string name; int course; double avg; };` і масив таких записів. Звертання до поля — через крапку: `s[i].avg`. Під час сортування масиву структур порівнюють потрібне поле, а обмінюють записи цілком.

Перелічувані типи замінюють «магічні числа» зрозумілими іменами: замість кодів 0, 1, 2 для статусу замовлення використовують `enum class Status { New, Paid, Shipped };`. Перевага `enum class` над звичайним `enum` — значення не змішуються з цілими числами та з іншими переліченнями.

Опрацюйте контейнер `vector` — динамічний масив, який сам змінює свій розмір. Методи `push_back` (додати в кінець), `pop_back`, `size`, `clear`,

`erase`; обхід — звичайним циклом з індексом або циклом `for (int x : v)`. Перепишіть одну з задач теми «Масиви» з використанням `vector` і порівняйте зручність. Ознайомтеся з `map` (пари «ключ — значення», наприклад «слово — кількість») та `set` (множина унікальних елементів, що автоматично впорядковується).

Бібліотека `<algorithm>` містить готові алгоритми, які замінюють ручні цикли: `sort(v.begin(), v.end())`, `reverse`, `count`, `find`, `max_element`; функція `accumulate` (бібліотека `<numeric>`) обчислює суму. Для сортування структур за полем передайте у `sort` функцію порівняння: `bool byAvg(const Student &a, const Student &b) { return a.avg > b.avg; }`.

ПРАКТИЧНЕ ЗАВДАННЯ

Розробити програму «Каталог книг»: структура `Book` (назва, автор, рік видання, ціна, жанр — `enum class`), дані зберігаються у `vector<Book>`. Реалізувати меню: додати книгу, вивести всі книги у вигляді таблиці, знайти книги заданого автора, вивести книги, видані після заданого року, відсортувати за ціною (за допомогою `sort` з функцією порівняння), обчислити середню ціну. Окремо за допомогою `map<string, int>` підрахувати, скільки разів кожне слово трапляється у введеному тексті, а за допомогою `set` вивести унікальні слова в алфавітному порядку.

Запитання для самоперевірки:

1. Чим структура відрізняється від масиву?
2. Які переваги має контейнер `vector` порівняно зі статичним масивом?
3. Як відсортувати вектор структур за значенням одного з полів?

ТЕМА 13. РОБОТА З ФАЙЛАМИ. ОПРАЦЮВАННЯ ВИКЛЮЧНИХ СИТУАЦІЙ. ТЕСТУВАННЯ ПРОГРАМ

Література: *2, 4; інформаційні ресурси: 5, 10, 19, 25.*

Кількість годин для самостійної роботи: *6.*

Питання для самостійного вивчення:

1. Запис результатів у текстовий файл і читання даних із файлу. Опрацювання CSV-файлів.
2. Запис і читання масиву структур у двійковому файлі.
3. Перевірка коректності введення та генерація виключень (try, catch, throw).
4. Складання тестових наборів: типові, межові та некоректні значення.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Для роботи з файлами підключають бібліотеку `<fstream>`: `ofstream` — для запису, `ifstream` — для читання. Після відкриття **завжди** перевіряйте успішність: `if (!fin) { cout << "Файл не знайдено"; return 1; }`. Файл даних має лежати в тій самій папці, що й файл програми `.cpp` (для проєкту Dev-C++ — у папці проєкту); в OnlineGDB його створюють кнопкою «+» поруч із `main.cpp`. Режим `ios::app` дописує дані в кінець файлу, не стираючи попередні.

Правильний цикл читання — `while (fin >> x)` або `while (getline(fin, line))`: цикл завершується, щойно читання стає неможливим. Конструкція `while (!fin.eof())` — типова помилка, через яку останнє значення опрацьовується двічі. CSV-файл — це текстова таблиця, де значення в рядку розділені комами або крапками з комою. Рядок розбирають за допомогою `stringstream` і `getline(ss, field, ';')`, а числа перетворюють функціями `stoi` та `stod`.

Двійкові файли зберігають дані в тому самому вигляді, що й у пам’яті. Структуру записують одним викликом: `fout.write(reinterpret_cast<const char*>(&rec), sizeof(rec));` і так само читають через `read`. Для двійкових

файлів поля структури мають бути фіксованого розміру (`char name[30]`, а не `string`).

Опрацюйте механізм виключень. Якщо функція виявила помилку, яку не може виправити сама, вона генерує виключення: `throw invalid_argument("Від'ємна сума");`. Код, що може спричинити помилку, розміщують у блоці `try`, а реакцію на помилку — у блоці `catch (const exception &e)`, де `e.what()` повертає текст повідомлення. Перевіряйте введення: якщо користувач ввів літери замість числа, потік `cin` переходить у стан помилки, який скидають `cin.clear()` і `cin.ignore(10000, '\n')`.

Навчіться тестувати власні програми. Для кожної програми складіть таблицю тестів: номер, вхідні дані, очікуваний результат, фактичний результат, висновок. Обов'язково включайте типові значення, **межові** (0, 1, найбільше допустиме) та **некоректні** (від'ємні, нечислові, відсутній файл).

ПРАКТИЧНЕ ЗАВДАННЯ

Розробити програму «Облік витрат»: витрати зберігаються у CSV-файлі `expenses.csv` (дата; категорія; сума). Програма читає файл (з перевіркою його наявності), виводить таблицю витрат, загальну суму та суму за кожною категорією, дозволяє додати нову витрату з перевіркою коректності (сума — додатне число, категорія — непорожня; у разі помилки генерується та перехоплюється виключення) і дописує її у файл. Зберегти масив витрат також у двійковий файл і прочитати його назад. Скласти таблицю з 8–10 тестів (типові, межові, некоректні дані) та записати результати їх виконання.

Запитання для самоперевірки:

1. Чому цикл `while (!fin.eof())` вважається помилковим способом читання файлу?
2. Як працює конструкція `try … catch` і для чого використовують `throw`?
3. Які групи тестових значень потрібно перевіряти під час тестування програми?

ТЕМА 14. ОСНОВИ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ. КЛАСИ ТА ОБ'ЄКТИ

Література: *1, 3, 4; інформаційні ресурси: 5, 10, 19.*

Кількість годин для самостійної роботи: *12.*

Питання для самостійного вивчення:

1. Перетворення структури на клас: закриті поля, методи доступу, валідація значень.
2. Конструктори (за замовчуванням, з параметрами, копіювання) та деструктор. Порядок створення й знищення об'єктів.
3. Статичні поля, константні методи. Композиція об'єктів.
4. Перевантаження операцій +, −, ==, =, << та >>. Розділення класу на файли .h та .cpp.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Клас — це «креслення» об'єкта, яке поєднує дані (поля) та дії над ними (методи). Об'єкт — конкретний екземпляр класу. Головна ідея **інкапсуляції**: поля оголошують у розділі `private`, а доступ до них надають через відкриті методи `get…` і `set…`. Метод `set` перевіряє коректність значення, тому об'єкт ніколи не опиниться в неправильному стані (наприклад, баланс рахунку не стане від'ємним). Візьміть структуру «Студент» з теми 12 і перетворіть її на клас із закритими полями та перевіркою значень.

Опрацюйте конструктори — спеціальні методи, які автоматично викликаються під час створення об'єкта й задають його початковий стан. Конструктор має ім'я класу і не має типу результату. Розрізняйте конструктор за замовчуванням, з параметрами та копіювання. Поля зручно ініціалізувати списком ініціалізації: `Point(int x, int y) : x(x), y(y) {}`. Деструктор (`~ІмʼяКласу()`) викликається під час знищення об'єкта; у ньому звільняють ресурси. Додайте в конструктори й деструктор виведення повідомлень і простежте, в якому порядку створюються та знищуються об'єкти.

Розберіться зі **статичними** членами класу: статичне поле спільне для всіх об’єктів (наприклад, лічильник створених об’єктів), а статичний метод можна викликати без об’єкта: `Account::getCount()`. Методи, які не змінюють об’єкт, позначайте словом `const` після списку параметрів. **Композиція** — включення об’єкта одного класу як поля іншого (клас `Order` містить об’єкт `Date`).

Перевантаження операцій дає змогу працювати з власними об’єктами так само, як із вбудованими типами: `Fraction c = a + b;`, `if (a == b)`, `cout << a;`. Операції `<<` і `>>` реалізують як дружні функції, що повертають посилання на потік. Для великих класів оголошення записують у файл `.h`, а реалізації методів — у `.cpp` з використанням операції `::` (`void Account::deposit(double s) { … }`).

ПРАКТИЧНЕ ЗАВДАННЯ

Розробити клас `Fraction` (звичайний дріб) із закритими полями «чисельник» і «знаменник» (знаменник не може дорівнювати нулю — перевірка в конструкторі та методі `set`), конструкторами за замовчуванням, з параметрами та копіювання, методом скорочення дробу, статичним лічильником створених об’єктів і перевантаженими операціями `+`, `-`, `*`, `==`, `<<`, `>>`. Розмістити клас у файлах `fraction.h` і `fraction.cpp`. У `main()` продемонструвати всі можливості класу; додати до конструкторів і деструктора виведення повідомлень та пояснити в коментарях порядок їх виклику.

Запитання для самоперевірки:

1. Що таке інкапсуляція і як вона реалізується в C++?
2. Коли викликаються конструктор і деструктор? Яке їхнє призначення?
3. Для чого перевантажують операції `<<` і `>>` і чому їх оголошують дружніми функціями?

ТЕМА 15. УСПАДКУВАННЯ ТА ПОЛІМОРФІЗМ

Література: *1, 3, 4; інформаційні ресурси: 5, 10, 19.*

Кількість годин для самостійної роботи: *8.*

Питання для самостійного вивчення:

1. Діаграми класів (UML) для предметної області.
2. Ієрархії класів. Порядок виклику конструкторів і деструкторів під час успадкування.
3. Перевизначення методів. Виклик через покажчик на базовий клас із virtual і без нього.
4. Абстрактні класи та чисто віртуальні методи. Проблема невіртуального деструктора.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Успадкування дає змогу створити новий клас на основі наявного: похідний клас отримує всі поля й методи базового та додає власні. Записують так: `class Circle : public Shape { … };`. Перш ніж писати код, побудуйте діаграму класів: кожен клас — прямокутник з трьома частинами (ім’я, поля, методи), стрілка з порожнім трикутником вказує від похідного класу до базового. Модифікатори доступу позначають символами: - private, # protected, + public.

Простежте порядок викликів: під час створення об’єкта похідного класу спочатку викликається конструктор базового класу, потім — похідного; деструктори — у зворотному порядку. Параметри базовому конструктору передають у списку ініціалізації: `Circle(double r) : Shape("Коло"), r(r) {}`. Додайте виведення повідомлень у конструктори і деструктори трирівневої ієрархії та перевірте це на практиці.

Головне питання теми — **поліморфізм**. Створіть масив покажчиків на базовий клас `Shape*`, заповніть його об’єктами різних похідних класів і викличте для кожного метод `area()`. Без слова `virtual` для всіх об’єктів буде викликано метод базового класу; з `virtual` — метод відповідного похідного

класу (пізнє зв’язування). Порівняйте обидва варіанти та запишіть висновок. Позначайте перевизначені методи словом `override` — компілятор перевірить, що метод справді перевизначає віртуальний.

Абстрактний клас містить хоча б один **чисто віртуальний** метод (`virtual double area() const = 0;`) і слугує лише основою для похідних класів — створити його об’єкт неможливо. Базовий клас ієрархії обов’язково повинен мати **віртуальний деструктор**; інакше під час `delete` через покажчик на базовий клас деструктор похідного класу не буде викликано, і ресурси не звільняться. Продемонструйте цю проблему та її виправлення.

ПРАКТИЧНЕ ЗАВДАННЯ

Розробити ієрархію класів «Співробітники»: абстрактний клас `Employee` (ім’я, стаж; чисто віртуальні методи `salary()` та `info()`; віртуальний деструктор) і похідні класи `Manager` (оклад + премія), `Engineer` (погодинна оплата), `Intern` (фіксована стипендія). Створити масив (або `vector`) покажчиків `Employee*` з об’єктами різних класів, вивести інформацію про кожного, загальний фонд заробітної плати та співробітника з найбільшою зарплатою; коректно звільнити пам’ять. Побудувати діаграму класів ієрархії та продемонструвати порядок виклику конструкторів і деструкторів.

Запитання для самоперевірки:

1. У якому порядку викликаються конструктори і деструктори базового та похідного класів?
2. Що змінюється у виклику методу через покажчик на базовий клас, якщо метод оголошено як `virtual`?
3. Для чого базовому класу потрібен віртуальний деструктор?

ТЕМА 16. ПРАКТИКУМ З ПРОГРАМУВАННЯ: РОЗРОБЛЕННЯ ТА ЗАХИСТ ПРИКЛАДНОГО ПРОЄКТУ

Література: *1, 2, 3, 4; інформаційні ресурси: 15, 16, 19, 25.*

Кількість годин для самостійної роботи: *8.*

Питання для самостійного вивчення:

1. Вибір теми індивідуального проєкту. Формулювання функціональних вимог і план роботи.
2. Проєктування структури програми: діаграма класів, схема меню, формат файлів даних.
3. Поетапна реалізація проєкту з використанням Git. Тестування та виправлення помилок.
4. Документування коду, інструкція користувача, підготовка презентації та захисту.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

Завершальна тема об’єднує все вивчене: змінні, розгалуження, цикли, масиви й рядки, функції, структури, файли, виключення, класи. Почніть із **постановки задачі**: оберіть тему (з переліку в ПР № 35 або власну, погоджену з викладачем) і письмово сформулюйте 6–10 функціональних вимог у форматі «Програма повинна …», а також вхідні та вихідні дані. Чітко сформульовані вимоги — половина успіху проєкту.

Спроєктуйте програму до написання коду. Визначте класи предметної області та зв’язки між ними (діаграма класів), опишіть пункти меню та формат файлу даних (наприклад, CSV з розділювачем `;`). Розбийте роботу на етапи: каркас меню → класи → збереження та завантаження даних → пошук і сортування → опрацювання помилок → тестування. Після завершення кожного етапу програма повинна компілюватися та запускатися.

Ведіть проєкт у Git: після кожного завершеного етапу виконуйте коміт зі змістовним повідомленням («Додано збереження у файл», «Виправлено помилку пошуку»), а наприкінці завантажте репозиторій на GitHub. Тестуйте

програму на типових, межових і некоректних даних і заносьте результати до таблиці тестів; знайдені помилки виправляйте й тестуйте повторно.

Завершіть роботу документуванням: коментар-заголовок у кожному файлі, коментарі до класів і функцій (призначення, параметри, результат), файл `README.md` з описом проєкту та інструкцією користувача. Підготуйте презентацію на 5–7 слайдів і доповідь на 5 хвилин: постановка задачі, структура класів, демонстрація роботи, результати тестування, висновки. Відрепетируйте захист і підготуйте відповіді на можливі запитання: чому обрано саме такі класи, як працює збереження у файл, як програма реагує на некоректні дані.

ПРАКТИЧНЕ ЗАВДАННЯ

Виконати індивідуальний проєкт за погодженою з викладачем темою (наприклад, «Облік товарів на складі», «Бібліотека», «Розклад занять», «Особистий бюджет», «Журнал успішності»). Програма повинна містити: меню з повторним виведенням до вибору виходу; не менше трьох класів, зокрема ієрархію з віртуальними методами; збереження та завантаження даних у файл формату CSV; пошук, фільтрацію та сортування записів; перевірку коректності введення та опрацювання виключень. Підготувати технічне завдання, діаграму класів, таблицю тестів (не менше 10 тестів), файл README.md, Git-репозиторій з історією комітів і презентацію для захисту.

Запитання для самоперевірки:

1. Назвіть основні етапи розроблення програмного продукту.
2. Чому тестування програми на межових і некоректних даних є обов’язковим?
3. Які елементи має містити документація програмного проєкту?

КРИТЕРІЇ ОЦІНЮВАННЯ ВИКОНАННЯ САМОСТІЙНОЇ РОБОТИ

Оцінювання результатів самостійної роботи студента здійснюється під час поточного контролю у формі перевірки виконаних практичних завдань (програмного коду та результатів його роботи), усного опитування за запитаннями для самоперевірки та перевірки конспектів опрацьованих тем.

Під час оцінювання враховується: повнота опрацювання теоретичних питань; працездатність і правильність розробленого програмного коду (програма компілюється без помилок і дає правильні результати на тестових даних); доцільність застосування вивчених мовних конструкцій; якість оформлення коду (змістовні імена, відступи, коментарі); самостійність виконання завдань; уміння пояснити логіку власного алгоритму та відповісти на запитання викладача.

Програмний код, який студент не може пояснити, або код, ідентичний роботам інших студентів, не зараховується. Практичне завдання з теми 16 (індивідуальний проєкт) оцінюється під час його захисту.

Наявність виконаної самостійної роботи за всіма темами (оформлених практичних завдань та відповідей на запитання для самоперевірки) є обов’язковим компонентом для отримання максимального поточного семестрового балу та допуску до підсумкового диференційованого заліку.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ ТА ІНФОРМАЦІЙНИХ РЕСУРСІВ

Основна література

1. Зеленський О. С., Лисенко В. С. Об’єктно-орієнтоване програмування на C++ : навч. посіб. Кривий Ріг : ДУЕТ, 2023. 215 с.
2. Ярошко С. А., Ярошко О. С. Методи розробки алгоритмів. Програмування мовою C++ : навч. посіб. Львів : ЛНУ імені Івана Франка, 2022. 248 с.

Додаткова література

3. Міловідов Ю. О. Об’єктно-орієнтоване програмування : навч. посіб. 3-тє вид., допов. і переробл. Київ : НУБіП України, 2025. 334 с.
4. Stroustrup B. Programming: Principles and Practice Using C++. 3rd ed. Boston : Addison-Wesley, 2024. 656 p.

Інформаційні ресурси

5. C++ Core Guidelines : вебсайт. URL: https://isocpp.github.io/CppCoreGuidelines/ (дата звернення: 11.09.2025).
6. C++ Insights : вебсайт. URL: https://cppinsights.io (дата звернення: 05.09.2025).
7. Codewars : вебсайт. URL: https://www.codewars.com (дата звернення: 22.10.2025).
8. Compiler Explorer (Godbolt) : вебсайт. URL: https://godbolt.org (дата звернення: 28.09.2025).
9. cplusplus.com : вебсайт. URL: https://cplusplus.com (дата звернення: 21.09.2025).
10. cppreference.com : вебсайт. URL: https://en.cppreference.com (дата звернення: 29.09.2025).
11. e-olymp : вебсайт. URL: https://www.e-olymp.com (дата звернення: 18.10.2025).
12. Exercism. C++ Track : вебсайт. URL: https://exercism.org/tracks/cpp (дата звернення: 13.10.2025).
13. freeCodeCamp : вебсайт. URL: https://www.freecodecamp.org (дата звернення: 03.11.2025).
14. GeeksforGeeks. C++ : вебсайт. URL: https://www.geeksforgeeks.org/c-plus-plus/ (дата звернення: 14.09.2025).
15. Git : вебсайт. URL: https://git-scm.com/doc (дата звернення: 09.09.2025).

16. GitHub : вебсайт. URL: https://github.com (дата звернення: 07.09.2025).
17. GNU Compiler Collection (GCC) : вебсайт. URL: https://gcc.gnu.org (дата звернення: 08.10.2025).
18. HackerRank. C++ : вебсайт. URL: https://www.hackerrank.com/domains/cpp (дата звернення: 17.09.2025).
19. LearnCpp.com : вебсайт. URL: https://www.learncpp.com (дата звернення: 24.10.2025).
20. LeetCode : вебсайт. URL: https://leetcode.com (дата звернення: 16.09.2025).
21. OnlineGDB : вебсайт. URL: https://www.onlinegdb.com (дата звернення: 05.11.2025).
22. Programiz. Learn C++ Programming : вебсайт. URL: https://www.programiz.com/cpp-programming (дата звернення: 07.10.2025).
23. Prometheus : вебсайт. URL: https://prometheus.org.ua (дата звернення: 07.11.2025).
24. Replit : вебсайт. URL: https://replit.com (дата звернення: 06.11.2025).
25. Stack Overflow : вебсайт. URL: https://stackoverflow.com (дата звернення: 06.09.2025).
26. Standard C++ Foundation : вебсайт. URL: https://isocpp.org (дата звернення: 02.11.2025).
27. Visual Studio Code : вебсайт. URL: https://code.visualstudio.com/docs (дата звернення: 09.11.2025).