ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ:

методичні рекомендації
до виконання індивідуальних завдань

Запоріжжя 2025

metinvest polytechnic

УДК 004.42(072)
О13

Рекомендовано Науково-методичною радою ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»
(протокол № 2 від 21.11.2025 р.)

Укладачі:
Нікуліна О.М., д-р. техн. наук, професор
Кондратов О.М., старший викладач

О13 **АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ :** методичні рекомендації до виконання індивідуальних занять / уклад. О. М. Нікуліна, О. М. Кондратов. Запоріжжя : ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», 2025. 54 с.

Методичні вказівки включають теми, мету, теоретичні основи, приклади програм та завдання для кожного індивідуального завдання, вимоги до оформлення звіту індивідуального завдання, список рекомендованих джерел.

Рекомендовано для студентів спеціальності F3 Комп'ютерні науки першого (бакалаврського) рівня освіти.

УДК 004.42(072)

ЗМІСТ

ВСТУП

Рекомендації призначені для студентів, які вивчають мову С++ на семінарах або самостійно.

Предметом навчальної дисципліни «Алгоритмізації та програмування» є методи алгоритмізації та програмування на мовах С++, С# та Java. Завдання дисципліни – освоєння методів та засобів процедурного та вступ до об’єктно-орієнтованого програмування у візуальному середовищах Visual Studio та IntelliJ IDEA.

Мета цих методичних рекомендацій – допомогти студентам виконати індивідуальні завдання з навчальної дисципліни «Алгоритмізації та програмування».

Індивідуальні завдання передбачають вивчення розділів: цикли, одновимірні масиви, функції, побудова та об’ява класів. Для виконання завдань студент повинен в достатньому ступені володіти процедурним програмуванням. До кожної індивідуального завдання в методичних рекомендаціях наведені: мета роботи, теоретичні основи, приклади та варіанти задач за розділами курсів, що вивчаються. Варіанти робіт відповідають вимогам навчального плану та силабусу і можуть бути застосовані для здійснення контролю знань студентів зі спеціальностями F3 – «Комп’ютерні науки».

Дисципліна спрямована на отримання здобувачами наступних загальних та спеціальних (фахових) компетентностей:

ЗК1. Здатність до абстрактного мислення, аналізу та синтезу.

ЗК2. Здатність застосовувати знання у практичних ситуаціях.

ЗК3. Знання та розуміння предметної області та розуміння професійної діяльності.

ЗК6. Здатність вчитися й оволодівати сучасними знаннями.

ЗК9. Здатність працювати в команді.

СК3 Здатність до логічного мислення, побудови логічних висновків, використання формальних мов і моделей алгоритмічних обчислень, проєктування, розроблення й аналізу алгоритмів, оцінювання їх ефективності та складності, розв’язності та нерозв’язності алгоритмічних проблем для адекватного моделювання предметних областей і створення програмних та інформаційних систем.

СК7 Здатність застосовувати теоретичні та практичні основи методології та технології моделювання для дослідження характеристик і поведінки складних об’єктів і систем, проводити обчислювальні експерименти з обробкою й аналізом результатів.

СК8. Здатність проєктувати та розробляти програмне забезпечення із застосуванням різних парадигм програмування: узагальненого, об’єктно-орієнтованого, функціонального, логічного, з відповідними моделями, методами й алгоритмами обчислень, структурами даних і механізмами управління.

Дисципліна спрямована на отримання здобувачами наступних програмних результатів :

ПР5. Проєктувати, розробляти та аналізувати алгоритми розв’язання обчислювальних та логічних задач, оцінювати ефективність та складність алгоритмів на основі застосування формальних моделей алгоритмів та обчислюваних функцій.

ПР9. Розробляти програмні моделі предметних середовищ, вибирати парадигму програмування з позицій зручності та якості застосування для реалізації методів та алгоритмів розв’язання задач в галузі комп’ютерних наук.

У результаті виконання практичних робіт здобувач вищої освіти повинен продемонструвати достатній рівень сформованості наступних результатів навчання:

- Здатність до логічного мислення, побудови логічних висновків, використання формальних мов і моделей алгоритмічних обчислень, проєктування, розроблення й аналізу алгоритмів, оцінювання їх ефективності та складності, розв’язності та нерозв’язності алгоритмічних проблем для адекватного моделювання предметних областей і створення програмних та інформаційних систем.
- Здатність застосовувати існуючі і розробляти нові алгоритми розв’язування задач у галузі комп’ютерних наук.
- Здатність розробляти і реалізовувати проекти зі створення програмного забезпечення, у тому числі в непередбачуваних умовах, за нечітких вимог та необхідності застосовувати нові стратегічні підходи, використовувати програмні інструменти для організації командної роботи над проєктом.
- Здатність проєктувати та розробляти програмне забезпечення із застосуванням різних парадигм програмування: узагальненого, об’єктно-орієнтованого, функціонального, логічного, з відповідними моделями, методами й алгоритмами обчислень, структурами даних і механізмів управління.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ 1

Робота зі статичними масивами на мові С++

Мета – опанувати написання програм з використанням статичних масивів на мові С++.

1.1 Теоретичні основи

Одновимірний масив описується вказанням типу елементів, імені масиву та кількості його елементів:

```
тип ім’я[розмір];
```

де тип – спільний для всіх елементів масиву; ім’я – назва масиву; розмір – кількість елементів масиву.

Важливо враховувати:

- кількість елементів масиву задається лише натуральною константою або додатним виразом із констант;
- параметр розмір має бути цілочислового типу;
- змінні не можна використовувати для задання кількості елементів;
- усі елементи масиву одного типу, який може бути будь-яким (наприклад, int, double, char, bool);
- елементи розміщуються послідовно в оперативній пам’яті;
- нумерація елементів починається з нуля;
- розмір масиву фіксований і не змінюється під час виконання програми.

Приклади опису масивів:

```
int a[20], b[100];
bool flags[10];
```

Для звернення до елемента масиву вказують ім’я масиву та індекс у квадратних дужках. Індекс визначає положення елемента всередині масиву, тобто його зсув відносно першого елемента, який має індекс 0. Індексування може здійснюватися константами, змінними або виразами, результат яких має бути дискретного (цілого) типу. При цьому компілятор автоматично перетворює значення до цілочислового типу. Допустимий діапазон індексу зазвичай не перевищує 2 Гбайт, що визначає максимально можливу кількість елементів у масиві. Це ж обмеження поширюється й на загальний обсяг пам’яті, яку займає масив у програмі.

Елементи масиву нумеруються від нуля, і програміст повинен

самостійно стежити, щоб індекс не виходив за допустимі межі. Якщо індекс набуває недопустимого значення (від'ємного або більшого за номер останнього елемента), компілятор цього не перевіряє, що може спричинити помилки під час виконання програми. У таких випадках програма може або аварійно завершитися, або – що ще небезпечніше – продовжити роботу з неправильними результатами, змінюючи випадкові області пам'яті. Тому важливо завжди контролювати коректність індексів у масивах, особливо при роботі з циклами або введенням даних користувачем.

Масиви можуть бути ініціалізовані при оголошенні, тобто отримати початкові значення елементів. Ініціалізація здійснюється переліком значень у фігурних дужках, розділених комами. У цьому випадку кожен елемент масиву отримує відповідне значення. Якщо вказані не всі значення, решта елементів автоматично ініціалізуються нулями. Коли ініціалізуються усі елементи, розмір масиву можна не вказувати, адже компілятор сам визначає його за кількістю поданих значень. Також можливо створювати частково ініціалізовані масиви або масиви з автоматично визначеним розміром, що зручно для компактного й безпечного оголошення даних.

Наприклад:

```
float F[4];           // Визначення без ініціалізації
int I1[3] = {1, 2, 3 }, I2[] = {11, 22, 33};   // Масиви із трьома
елементами
long int LI[10] = {7, 2, 17};   // Значення отримують
                          // тільки перші три елементи
```

Розмір масиву може задаватися також за допомогою специфікатора `typedef` попередньо описати масивний тип, а потім вживати його для опису масивів:

```
const int
  N = 100;
Typedef
  double TDbArray[N];                    // Тип – масив
                  // Можливі такі методи опису масиву
  double A1[2 * N], A2[N];
  TDbArray A3;
```

Операція `sizeof`, застосована до імені масиву або імені масивного типу, повертає кількість байт, що відводиться під масив. Наприклад, для наведених вище описів, можливе виконання таких операторів:

```
int s1 = sizeof (A1);                    // s1 == 1600
int s2 = sizeof A2;                      // s2 == 800
int s3 = sizeof(TDbArray);               // s3 == 800
```

Внутрішня організація масивів у пам'яті. Усі елементи масиву розміщуються послідовно в оперативній пам'яті. Якщо **`addr_0`** — адреса першого елемента, то адресу будь-якого іншого елемента можна обчислити за формулою:

```
addr_i = addr_0 + i * sizeof(елемент)
```

де `sizeof(елемент)` — кількість байтів, яку займає один елемент.

Наприклад, якщо кожне число типу int займає 4 байти, то:

```
addr_0 – адреса a[0]
addr_1 = addr_0 + 4 – адреса a[1]
addr_2 = addr_0 + 8 – адреса a[2]
```

Завдяки такій організації доступ до будь-якого елемента відбувається швидко, без послідовного пошуку.

Оператор sizeof дозволяє визначити обсяг пам'яті, який займає масив у байтах, а також дізнатися розмір одного елемента. Наприклад:

```
int arr[10];
cout << sizeof(arr);    // 40, якщо sizeof(int) == 4
cout << sizeof(arr[0]); // 4
```

Знаючи ці два значення, можна легко знайти кількість елементів у масиві:

```
int n = sizeof(arr) / sizeof(arr[0]); // n == 10
```

Типові помилки при роботі зі статичними масивами:

- Вихід за межі масиву — найчастіша помилка, яка призводить до непередбачуваної поведінки програми.
- Використання неініціалізованих елементів — може дати випадкові значення.
- Копіювання масивів через оператор = — у C++ так робити не можна:

```
int a[3] = {1, 2, 3}, b[3];
b = a; // Помилка!
```

Для копіювання потрібно використовувати цикл або функцію `memcpy`

Переваги та обмеження статичних масивів

Переваги:

- висока швидкість доступу до елементів;
- простота реалізації;

- ефективне використання пам’яті.

Обмеження:

- розмір масиву не можна змінити після створення;
- компілятор не перевіряє вихід за межі;
- незручність при роботі зі змінними обсягами даних.

Багатовимірні статичні масиви в мові C++. Мова C++ дозволяє створювати масиви з більш ніж одним виміром. Такі масиви називають багатовимірними. Кожен вимір визначається окремим індексом. Найчастіше використовуються двовимірні масиви, які зручно уявляти як таблицю або матрицю з рядків і стовпців.

Створення двовимірного масиву. Оголошення двовимірного масиву має вигляд:

```
тип_елементів ім’я_масиву[кількість_рядків][кількість_стовпців];
```

Наприклад:

```
const int m = 3;
const int n = 4;
double a[m][n];
```

Цей масив складається з `m` рядків і `n` стовпців, тобто загалом має `mxn` елементів.

Перший індекс зазвичай відповідає номеру рядка, а другий — номеру стовпця.

Внутрішня організація в пам’яті. Фізично двовимірний масив зберігається в пам’яті послідовно. Елементи записуються рядок за рядком, тобто спочатку всі елементи першого рядка, потім другого тощо. Наприклад, масив:

```
double a[3][4];
```

у пам’яті буде розміщено як:

```
a[0][0], a[0][1], a[0][2], a[0][3],
a[1][0], a[1][1], a[1][2], a[1][3],
a[2][0], a[2][1], a[2][2], a[2][3]
```

Отже, фактично це одновимірний блок пам’яті, який містить `m * n` послідовних елементів.

Той самий масив можна оголосити й так:

```
double b[m * n];
b[i * n + j] = 0;  // еквівалентно a[i][j] = 0;
```

Однак використання двовимірного синтаксису зручніше і зрозуміліше для програміста.

Ініціалізація двовимірних масивів. Багатовимірні масиви можна ініціалізувати під час оголошення. Наприклад:

```
int theArray[5][3] = {
    1, 2, 3,
    4, 5, 6,
    7, 8, 9,
    10, 11, 12,
    13, 14, 15
};
```

Це означає, що перші три елементи запишуться в рядок з індексом 0, наступні три — у рядок 1 і так далі.

Для кращої наочності ініціалізацію можна оформити вкладеними фігурними дужками:

```
int theArray[5][3] = {
    { 1,  2,  3 },
    { 4,  5,  6 },
    { 7,  8,  9 },
    {10, 11, 12 },
    {13, 14, 15 }
};
```

Внутрішні дужки використовуються лише для зручності читання — компілятор їх ігнорує.

Якщо вказати лише другу розмірність, першу компілятор обчислить сам:

```
int theArray[][3] = {
    { 1,  2,  3 },
    { 4,  5,  6 },
    { 7,  8,  9 }
};
```

Елементи, для яких не вказано значення, автоматично заповнюються нулями:

```
const int m = 4;
const int n = 3;
int arr[m][n] = { };  // Усі елементи дорівнюють 0
```

Доступ до елементів. Для звертання до конкретного елемента двовимірного масиву використовуються два індекси:

```
arr[рядок][стовпець]
```

Наприклад:

```
cout << arr[0][2]; // Елемент першого рядка, третього стовпця
```

Обхід двовимірного масиву. Для роботи з усіма елементами масиву зазвичай застосовують вкладені цикли:

```
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        arr[i][j] = i + j;
    }
}
```

Традиційно змінну і використовують для рядків, а j — для стовпців.

Цикли на діапазоні (range-based for). У сучасному C++ (починаючи з C++11) двовимірні масиви можна зручно обробляти за допомогою циклів на діапазоні:

```
for (auto& row : arr) {
    for (int& item : row) {
        item = 1;
    }
}
```

Так можна також вивести елементи таблиці на екран:

```
for (const auto& row : arr) {
    for (const int& item : row) {
        cout << item << "\t";
    }
    cout << endl;
}
```

Використання `auto` спрощує синтаксис, оскільки типовий заголовок циклу виглядав би складніше:

```
for (const int (&row)[n] : arr)
```

Передавання двовимірних масивів у функції

Передати багатовимірний масив у функцію можна кількома способами.

Найпростіший — явно вказати другу розмірність:

```
double f(double a[][3]) {
    return a[1][1];
}

int main() {
    double arr[][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    cout << f(arr);
}
```

У цьому випадку компілятор повинен знати другу розмірність масиву, інакше він не зможе правильно обчислити зміщення елементів у пам’яті.

Передавання за посиланням. Ще один зручний спосіб — передати двовимірний масив за посиланням, зберігаючи інформацію про обидва розміри:

```
#include <iostream>
using namespace std;

const int m = 2;
const int n = 3;

int sum(int (&arr)[m][n]) {
    int result = 0;
    for (const auto& row : arr) {
        for (const int& item : row) {
            result += item;
        }
    }
    return result;
}

int main() {
    int a[m][n] = {
        {1, 2, 4},
        {8, 16, 32}
    };
    cout << sum(a) << endl; // 63
}
```

Як і для одновимірних масивів, така функція працює лише з масивами визначених розмірів.

Особливості багатовимірних статичних масивів:

Переваги:

- простота доступу до елементів за двома або більше індексами;
- логічне представлення таблиць і матриць;
- швидкий доступ до елементів завдяки послідовному розміщенню в пам’яті.

Недоліки:

- фіксований розмір (визначається під час компіляції);
- обмежена гнучкість порівняно з динамічними структурами;
- передавання у функції потребує точного зазначення розмірів.

Опис функцій. Функція складається з заголовка і тіла:

```
тип ім’я_функції(список_параметрів)
{
    // тіло функції
}
```

Тип функції визначає тип значення, яке вона повертає.

Якщо функція не повертає результату — вказується тип void

Передача масивів у функції. Передача одновимірного масиву як параметра `(void f(int arr[], int n)` або `void f(int* arr, int n))`, використання посилань `(void f(int (&arr)[10]))`, обмеження компілятора (потрібно вказувати розмірність, крім першої).

Використання `const` для масивів. Як захистити масив від змін у функціях:

```
void printArray(const int arr[], int n);
```

Особливо важливо при передачі великих таблиць або константних даних.

Ініціалізація та заповнення масивів. Розширити те, що вже є:

- Ініціалізація часткова і повна;
- Ініціалізація масивів структур або рядків;
- Використання циклів для заповнення масиву (наприклад, обчислювальними формулами);
- Ініціалізація нулем через {0}.

Символьні (`char`) масиви і рядки. Символьні масиви як спосіб зберігання тексту до появи `std::string`. Правильне завершення нуль-символом '\0'.

Приклад:

```
char name[] = "Hello";
```

Відмінність між `char name[6] = "Hello"; і char* name = "Hello";`.

1.2. Приклади

Приклад 1.1

Упорядкувати числовий масив, що вміщує не більш 20 дійсних чисел, за неспаданням методом «бульбашки».

```
#include <iostream>
using namespace std;

int main()
{
    const int MAX_SIZE = 20;        // Максимальна кількість елементів
    double a[MAX_SIZE];             // Масив дійсних чисел
    int N;                          // Кількість елементів масиву
```

```
    // --- Введення розміру масиву ---
    cout << "Enter the number of elements (1-20): ";
    cin >> N;

    if (N < 1 || N > MAX_SIZE) {
        cout << "Error: number of elements must be between 1 and
20.\n";
        return 1;
    }

    // --- Введення елементів масиву ---
    cout << "\nEnter array elements:\n";
    for (int i = 0; i < N; i++) {
        cout << "a[" << i << "] = ";
        cin >> a[i];
    }

    // --- Виведення початкового масиву ---
    cout << "\nInitial array:\n";
    for (int i = 0; i < N; i++)
        cout << a[i] << " ";
    cout << endl;

    // --- Сортування методом бульбашки ---
    for (int i = 0; i < N - 1; i++) {
        for (int j = 0; j < N - i - 1; j++) {
            if (a[j] > a[j + 1]) {
                swap(a[j], a[j + 1]);
            }
        }
    }

    // --- Виведення відсортованого масиву ---
    cout << "\nSorted array (ascending order):\n";
    for (int i = 0; i < N; i++)
        cout << a[i] << " ";
    cout << endl;
}
```

Приклад 1.2

Дано натуральне число n ($n \le 20$), дійсне число w і масив з n дійсних чисел. Видалити з масиву елемент, що є найближчим до числа w.

```
#include <iostream>
#include <cmath>     // для функції fabs()
using namespace std;
```

```
int main()
{
    const int MAX_SIZE = 20;    // Максимальна кількість елементів у
масиві
    double arr[MAX_SIZE];       // Масив дійсних чисел
    double w;                   // Задане число
    int n;                      // Кількість елементів у масиві

    cout << "Enter w: ";
    cin >> w;
    cout << "Enter number of elements (n): ";
    cin >> n;

    // --- Введення елементів масиву ---
    for (int i = 0; i < n; i++)
    {
        cout << "arr[" << i << "] = ";
        cin >> arr[i];
    }

    // --- Вивід початкового масиву ---
    cout << "\nInitial array:\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    // --- Пошук елемента, найближчого до w ---
    int indexToRemove = 0;                  // Номер елементу, який
потрібно видалити
    double minDistance = fabs(w - arr[0]); // Мінімальна відстань до w

    for (int i = 1; i < n; i++)
    {
        double distance = fabs(w - arr[i]); // Поточна відстань
        if (distance < minDistance)
        {
            minDistance = distance;
            indexToRemove = i;              // Запам'ятовуємо позицію
ближчого елемента
        }
    }

    // --- Видалення знайденого елемента ---
    for (int i = indexToRemove + 1; i < n; i++)
        arr[i - 1] = arr[i]; // Зсуваємо елементи вліво
```

```
    n--; // Зменшуємо розмір масиву

    // --- Вивід результату ---
    cout << "\nResulting array";
    if (n == 0)
        cout << " is empty.";
    else
    {
        cout << ":\n";
        for (int i = 0; i < n; i++)
            cout << arr[i] << " ";
    }

    cout << endl;
}
```

Приклад 1.3

Дано натуральне число n ($n \le 100$) і масив з n дійсних чисел. Визначити в цьому масиві кількість ділянок, на яких його елементи убувають.

```
#include <iostream>

using namespace std;
int main()
{
    int n, i;                    // Кількість елементів масиву
    double a[100];                                    // Масив
    int c;                                // Кількість ділянок
    bool flag;
    cout << "n = ";
    cin >> n;
    for (i = 0; i < n; i++)
    {
        cout << "a[" << i << "] = ";
        cin >> a[i];
    }
    c = 0;
    flag = true;
    for (i = 1; i < n; i++)    // Починаємо з другого елементу
        if (a[i] > a[i - 1])                      // Є зростання
        {
            if (flag)                          // Нова ділянка
            {
                c++;
                flag = false;
```

```
            }
        }
        else                                    // Незростання
            flag = true;
    cout << "Count = " << c << '\n';
}
```

1.3. Задачі

Варіант 1

І. Дано натуральне число *n* і дійсне число *x*. Обчислити значення $\frac{x^{n^2}}{2^n}$.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

ІІ. Дано натуральне число *n* і цілочислова матриця $[a_{ij}]_{i,j=1,\ \dots\ ,n}$. Отримати $b_1, b_2, \dots, b_n$, де b_i – це $\max\limits_{1\le j\le n} a_{ij} \min\limits_{1\le j\le n} a_{ji}$. Розв'язати задачу, користуючись статичними масивами.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

Варіант 2

І. Дано натуральне число *n.* Обчислити

$$\frac{1}{\sin 1}+\frac{1}{\sin 1+\sin 2}+\dots+\frac{1}{\sin 1+\dots+\sin n}.$$

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

ІІ. Дано натуральне число *n.* За допомогою двовимірного дійсного числового масиву $[x_{ij}]_{i=1,2;\ j=1,\dots,n}$ на площині задано *n* точок так, що x_{1j}, x_{2j} – координати *j*-ї точки. Які дві з цих точок є найбільш близькими? Розв'язати задачу, користуючись статичним масивом.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

Варіант 3

I. Дано натуральне число *n*. Обчислити $\underbrace{\sqrt{1+\sqrt{2+\ldots+\sqrt{n}}}}_{n \text{ коренів}}$;

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

II. Дано масив дійсних чисел розміру $m \times n$, де m і n – натуральні числа. Поміняти в масиві місцями рядок з найбільшою і рядок з найменшою кількістю різних елементів. Розв'язати задачу, користуючись статичним масивом.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

Варіант 4

I. Дано натуральне число *n.* Обчислити $\frac{\cos 1}{\sin 1} \cdot \frac{\cos 1+\cos 2}{\sin 1+\sin 2} \cdot \ldots \cdot \frac{\cos 1+\ldots+\cos n}{\sin 1+\ldots+\sin n}$.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

II. Дано натуральне число *n.* За допомогою двовимірного дійсного числового масиву $[x_{ij}]_{i=1,2;\ j=1,\ldots,n}$ на площині задано n точок так, що x_{1j}, x_{2j} – координати j-ї точки. Знайти трикутник найбільшої площі з вершинами в цих точках. Розв'язати задачу, користуючись статичним масивом.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

Варіант 5

I. Дано дійсне число *a* і натуральне число *n*. Обчислити

$$\frac{1}{a}+\frac{1}{a(a+1)}+\ldots+\frac{1}{a(a+1)\ldots(a+n)}.$$

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

II. Дано натуральне число n ($n \geq 2$) і дійсний квадратний масив розміру $n \times n$. Побудувати послідовність $b_1, b_2, ..., b_n$ з нулів і одиниць, у якій $b_i = 1$ тоді і тільки тоді, коли елементи i-го рядка матриці утворюють зростаючу послідовність. Розв'язати задачу, користуючись статичними масивами.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

Варіант 6

I. Дано дійсне число a *і* натуральне число n. Обчислити $a(a-n)(a-2n)...(a-n^2)$.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

II. Дано натуральне число n і дійсний квадратний масив розміру $n \times n$. Розглянемо ті елементи, які розташовані в рядках, що починаються з від'ємного елементу. Знайти суми тих з них, які розташовані відповідно нижче, вище і на головній діагоналі. Розв'язати задачу, користуючись статичним масивом.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

Варіант 7

I. Дано натуральне число n. Отримати нове число, переставивши в представленні числа n першу і останню цифри.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

II. Дано натуральне число n. У заданому квадратному цілочисловому масиві розміру $n \times n$ визначити індекси всіх елементів з

максимальним значенням. Сформувати масив з цих індексів. Розв'язати задачу, користуючись статичними масивами.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

Варіант 8

I. Нехай $v_1 = v_2 = 0$; $v_3=1.5$; $v_i = \frac{i+1}{i^2+1} v_{i-1} - v_{i-2} v_{i-3}$, $i = 4, 5, \ldots$

Дано натуральне n ($n \geq 4$). Отримати v_n.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

II. Дано натуральні числа n і m. У заданому дійсному масиві розміру $m \times n$ поміняти місцями рядок, що містить елемент з найменшим значенням, і рядок, що містить елемент найбільш близький до середнього арифметичного всіх елементів масиву. Припускається, що ці елементи єдині. Розв'язати задачу, користуючись статичним масивом.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

Варіант 9

I. Дано дійсні числа a, h, натуральне число n. Обчислити

$f(a)+2f(a+h)+2f(a+2h)+\ldots+ 2f(a+(n-1)h)+f(a+nh)$,

де $f(x)=(x^2+1)\cos^2 x$

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

II. Дано квадратний масив. Повернути на 180° навколо вертикальної осі вміст верхнього і нижнього трикутників між двома його діагоналями. Фрагменти діагоналей при цьому не зачіпати. Розв'язати задачу, користуючись статичним масивом.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

Варіант 10

I. Дано дійсне число *x i* натуральне непарне число *n*. Обчислити

$$x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\frac{x^9}{9!}-\frac{x^{11}}{11!}+\cdots\pm\frac{x^n}{n!}.$$

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

II. Дано натуральне число *n* і цілочисловий квадратний масив розміру $n \times n$. Знайти номери рядків, елементи кожного з яких утворюють монотонну послідовність (монотонно спадну або монотонно зростаючу). Сформувати з цих рядків новий масив. Розв'язати задачу, користуючись статичними масивами.

1. Проаналізувати постановку задачі.

2. Обґрунтувати використовувані структури даних і алгоритм розв'язання задачі.

3. Написати і налаштувати програму для розв'язання задачі.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ 2

Робота з динамічними масивами та функціями на мові С++

Мета – опанувати написання програм з використанням динамічних масивів та функцій.

2.1. Теоретичні основи

У мові С++ програміст має змогу самостійно керувати пам’яттю за допомогою операторів `new` та `delete`. Це дозволяє створювати змінні та масиви довільного розміру, який визначається під час виконання програми, а не під час компіляції, як у випадку статичних масивів.

Для виділення пам’яті під окрему змінну використовують оператор new:

```
int* p = new int;  // виділення пам'яті для одного цілого числа
*p = 42;           // присвоєння значення
```

Змінна, яка була створена в динамічній пам'яті, повинна бути звільнена за допомогою оператору `delete`:

```
delete p;
```

Після звільнення пам’яті покажчик варто обнулити:

```
p = nullptr;
```

щоб уникнути «висячих» вказівників, які можуть призвести до збоїв програми.

Для створення масиву в динамічній пам’яті після `new` ставляться квадратні дужки з кількістю елементів:

```
int n;
cin >> n;
double* arr = new double[n];
```

Тут `n` може бути змінною, тому розмір динамічного масиву можна задавати під час виконання програми. Це основна перевага динамічних масивів порівняно зі статичними.

Доступ до елементів динамічного масиву здійснюється так само, як і до звичайного:

```
for (int i = 0; i < n; i++)
    arr[i] = i * 2;
```

Звільнення пам’яті для масиву виконується за допомогою

оператора:

```
delete[] arr;
```

Заборонено використовувати `delete` замість `delete[]`, інакше можливі помилки в роботі програми.

Багатовимірні динамічні масиви. Динамічні багатовимірні масиви реалізуються через масиви вказівників. Наприклад, створення двовимірного масиву розміру `m×n`:

```
int m, n;
cin >> m >> n;
double** a = new double*[m];   // масив вказівників
for (int i = 0; i < m; i++)
    a[i] = new double[n];      // створення кожного рядка
```

Заповнення і виведення:

```
for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
        a[i][j] = i + j;
```

Пам’ять потрібно звільняти в зворотному порядку:

```
for (int i = 0; i < m; i++)
    delete[] a[i];
delete[] a;
```

Аналогічним чином можна створювати тривимірні або багатовимірні масиви довільної розмірності.

Використання функцій для роботи з динамічними масивами. Функції в C++ — це незалежні програмні одиниці, які дозволяють розділяти програму на логічні частини.

У контексті динамічних масивів вони особливо корисні для:

- створення масиву;
- заповнення значеннями;
- обчислення статистичних характеристик (сума, середнє тощо);
- виведення масиву на екран;
- звільнення пам’яті.

Приклади роботи з динамічними масивами у функціях. Створення масиву:

```
double* createArray(int n) {
    return new double[n];
}
```

Заповнення масиву:

```
void fillArray(double* arr, int n) {
    for (int i = 0; i < n; i++)
        arr[i] = i * 1.5;
}
```

Виведення масиву:

```
void printArray(const double* arr, int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}
```

Звільнення пам’яті:

```
void deleteArray(double* arr) {
    delete[] arr;
}
```

Передача динамічних масивів у функції. Усі масиви в C++ передаються у функцію через вказівник на перший елемент. Це означає, що зміни, зроблені у функції, відображаються на вихідному масиві. Щоб захистити масив від змін, параметр можна оголосити як `const`:

```
void show(const int* a, int n);
```

Приклад виклику:

```
int n = 5;
int* a = new int[n];
fillArray(a, n);
show(a, n);
delete[] a;
```

Передача параметрів і область видимості. Формальні параметри функції - це локальні змінні, які створюються при виклику функції. Всі змінні, оголошені всередині функції, недоступні поза нею. Якщо потрібно, щоб функція змінювала значення змінної або вказівника, його передають за посиланням або через покажчик. Типові помилки при використанні динамічної пам’яті:

Забуте звільнення пам’яті (`delete` або `delete[]`): витік пам’яті (memory leak).

Подвійне звільнення тієї самої області пам’яті: недопустима операція.

Звернення до пам’яті після `delete`: висячий покажчик.

Вихід за межі масиву: непередбачувана поведінка програми.

Для уникнення таких помилок рекомендується після звільнення пам’яті обнуляти вказівники:

```
delete[] arr;
arr = nullptr;
```

Використання функцій для багатовимірних динамічних масивів. Так само можна передавати вказівники на вказівники:

```
void fillMatrix(double** a, int m, int n) {
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            a[i][j] = i + j;
}
```

Ці функції можна об’єднувати для створення, заповнення, виведення і звільнення матриць.

Відмінності між статичними та динамічними масивами зображено в таблиці 1.

Таблиця 1 - Відмінності між статичними та динамічними масивами

Ознака	**Статичний масив**	**Динамічний масив**
Місце зберігання	Стек (`stack`)	Куча (`heap`)
Задання розміру	На етапі компіляції	Під час виконання
Зміна розміру	Неможлива	Можлива (через перевиділення)
Звільнення пам’яті	Автоматично	Ручне (`delete[]`)
Швидкість доступу	Трохи вища	Залежить від виділення пам’яті

Перевиділення (`Reallocation`) динамічного масиву. Динамічний масив не можна змінити «на місці», тому для збільшення розміру доводиться створювати новий масив, копіювати дані та звільняти старий.

```
int* resizeArray(int* arr, int oldSize, int newSize) {
    int* newArr = new int[newSize];
    for (int i = 0; i < oldSize; i++)
        newArr[i] = arr[i];
    delete[] arr;
    return newArr;
}
```

Ця операція часто використовується у власних реалізаціях класів типу `vector`.

Динамічні масиви і стандартні контейнери. Використання `std::vector` замість «ручних» динамічних масивів полегшує роботу та автоматично керує пам'яттю:

```
#include <vector>
std::vector<int> v(n);
v.push_back(5);
```

Вектори також підтримують багатовимірність через вектор векторів (`std::vector<std::vector<int>>`).

Багатовимірні динамічні масиви через один блок пам'яті. Можна виділити однорідну область пам'яті для всієї матриці і обчислювати індекси вручну:

```
double* matrix = new double[m * n];
matrix[i * n + j] = value;
```

Перевага: менше виділень пам'яті, краще кешування. Динамічні масиви структур або об'єктів. Можна створювати масиви об'єктів класів:

```
class Point { public: int x, y; };
Point* points = new Point[n];
delete[] points;
```

Потрібно пам'ятати про виклики конструкторів і деструкторів.

Використання розумних покажчиків (`smart pointers`). `std::unique_ptr` або `std::shared_ptr` можуть автоматично звільняти пам'ять:

```
#include <memory>
std::unique_ptr<int[]> arr(new int[n]);
```

Це зменшує ймовірність витоків пам'яті.

Передача динамічних масивів у функції

Крім передачі через `int*`, можна передавати через `std::span` (C++20):

```
#include <span>
void printArray(std::span<int> arr) {
    for (int x : arr) std::cout << x << " ";
}
```

Не потрібно передавати розмір окремо, `span` зберігає його.

Помилки та відладка. Важливо контролювати:

- втечу пам'яті (не звільнена пам'ять);

- висячі покажчики;
- подвійне видалення;
- вихід за межі масиву;
- некоректне виділення для багатовимірних масивів.

2.2. Приклади

Приклад 2.1

Функція, яка визначає просте число.

```
bool proste(int n)
{
    bool f = true;
    for(int d = 2; d <= n /2; d++)
        if (n % d == 0)
        {
            f = false;
            break;
        }
    return f;
}
```

Приклад 2.2

Робота з динамічним матрицями.

```
#include <iostream>

using namespace std;

void input(int** a, int n);
void print(int** a, int n);
void peretvor(int** a, int n);
void in_n(int& n);
bool proste(int n);

int main()
{
    int** a, n;
    in_n(n);
    a = new int* [n];
    for (int i = 0; i < n; i++)
        a[i] = new int[n];
    input(a, n);
    print(a, n);
    peretvor(a, n);
    cout << endl;
    print(a, n);
```

```
    for (int i = 0; i < n; i++)
        delete a[i];
    delete[] a;
}

void in_n(int& n)
{
    cout << "n = \n";
    cin >> n;
}

void input(int** a, int n)
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            a[i][j] = rand() % 10;
}

void print(int** a, int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            cout << a[i][j] << "\t";
        cout << endl;
    }
}

bool proste(int n)
{
    bool f = true;
    for(int d = 2; d <= n /2; d++)
        if (n % d == 0)
        {
            f = false;
            break;
        }
    return f;
}

void peretvor(int** a, int n)
{
    for (int j = 0; j < n; j++)
    {
        if ((j + 1) % 2 == 0)
        {
```

```
            int sum = 0;
            for (int i = 0; i < n; i++)
                if (proste(a[i][j]))
                    sum += a[i][j];
            a[0][j] = sum;
        }
        else
        {
            int dob = 1;
            for (int i = 0; i < n; i++)
                if (proste(a[i][j]) == false)
                    dob *= a[i][j];
            a[0][j] = dob;
        }
    }
}
```

2.3. Задачі

Варіант 1

Розв'язати задачу з п. II ІДЗ 1 ІДЗ 1 з обов'язковим використанням динамічної пам'яті та покажчиків.

1. Визначити функції пошуку максимального елемента стовпця і мінімального елемента рядка двовимірного масиву.
2. Обґрунтувати перелік параметрів і тип функції.
3. Написати і налаштувати програму для розв'язання задачі

Варіант 2

Розв'язати задачу з п. II ІДЗ 1 з обов'язковим використанням динамічної пам'яті та покажчиків.

1. Визначити функцію пошуку двох найбільш близьких з n точок.
2. Обґрунтувати перелік параметрів і тип функції.
3. Написати і налаштувати програму для розв'язання задачі.

Варіант 3

Розв'язати задачу з п. II ІДЗ 1 з обов'язковим використанням динамічної пам'яті та покажчиків.

1. Визначити функцію обчислення кількості різних елементів в рядку двовимірного масиву.
2. Обґрунтувати перелік параметрів і тип функції.
3. Написати і налаштувати програму для розв'язання задачі.

Варіант 4

Розв’язати задачу з п. ІІ ІДЗ 1 з обов’язковим використанням динамічної пам’яті та покажчиків.

1. Визначити дві функції – обчислення площі трикутника за координатами його вершин і відшукання трикутника найбільшої площі серед всіх трикутників з вершинами, заданими двовимірним масивом.
2. Обґрунтувати перелік параметрів і тип функції.
3. Написати і налаштувати програму для розв’язання задачі.

Варіант 5

Розв’язати задачу з п. ІІ ІДЗ 1 з обов’язковим використанням динамічної пам’яті та покажчиків.

1. Визначити функцію перевірки рядка двовимірного масиву на зростання його елементів і функцію формування вказаного в умові масиву.
2. Обґрунтувати перелік параметрів і тип функції.
3. Написати і налаштувати програму для розв’язання задачі.

Варіант 6

Розв’язати задачу з п. ІІ ІДЗ 1 з обов’язковим використанням динамічної пам’яті та покажчиків.

1. Визначити функції обчислення кожної із зазначених трьох сум.
2. Обґрунтувати перелік параметрів і тип функції.
3. Написати і налаштувати програму для розв’язання задачі.

Варіант 7

Розв’язати задачу з п. ІІ ІДЗ 1 з обов’язковим використанням динамічної пам’яті та покажчиків.

1. Визначити функцію відшукання місця знаходження найбільшого елемента.
2. Обґрунтувати перелік параметрів і тип функції.
3. Написати і налаштувати програму для розв’язання задачі.

Варіант 8

Розв’язати задачу з п. ІІ ІДЗ 1 з обов’язковим використанням динамічної пам’яті та покажчиків.

1. Визначити функції пошуку місця розташування шуканих елементів і обчислення середнього арифметичного.
2. Обґрунтувати перелік параметрів і тип функції.
3. Написати і налаштувати програму для розв’язання задачі.

Варіант 9

Розв’язати задачу з п. ІІ ІДЗ 1 з обов’язковим використанням динамічної пам’яті та покажчиків.

1. Визначити функцію повороту заданого «кільця».
2. Обґрунтувати перелік параметрів і тип функції.
3. Написати і налаштувати програму для розв’язання задачі.

Варіант 10

Розв’язати задачу з п. ІІ ІДЗ 1 з обов’язковим використанням динамічної пам’яті та покажчиків.

1. Визначити функції перевірки рядка двовимірного масиву на його монотонне зростання або спадання і функцію формування необхідного масиву.
2. Обґрунтувати перелік параметрів і тип функції.
3. Написати і налаштувати програму для розв’язання задачі.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ 3
Створення класів на мові C++

Мета – опанувати написання програм з використанням класів.

3.1. Теоретичні основи [5]

У мові C++ класи є основним засобом для створення користувацьких типів даних, що дозволяє поєднувати дані і функції для роботи з ними в одному об’єкті. Клас (`class`) – це структурований тип даних, який включає поля (члени даних) та методи (функції-елементи), що описують поведінку об’єкта. Поля класу зберігають властивості об’єкта, а методи визначають дії над цими властивостями. Клас має ім’я, яке використовується для створення змінних типу цього класу – об’єктів або екземплярів класу.

Синтаксис опису класу має наступний вигляд:

```
class Name {
private:
// закриті поля і методи класу
protected:
// захищені поля і методи
public:
// публічні поля і методи
};
```

Одним з основних принципів об’єктно-орієнтованого програмування є інкапсуляція, тобто приховування даних. Інкапсуляція дозволяє обмежити доступ до внутрішніх елементів об’єкта і захистити їх від випадкових змін ззовні. У C++ реалізацію інкапсуляції забезпечують рівні доступу: `private`, `protected`, `public`. Поля та методи, оголошені як `private`, доступні лише всередині класу або у його друзів (класів та функцій). Елементи, оголошені як `protected`, додатково доступні в похідних класах. Елементи, оголошені як `public`, доступні будь-де у програмі.

Для доступу до закритих полів класу використовують спеціальні методи – геттери та сеттери (`getters` / `setters`). Вони дозволяють отримати значення поля або змінити його, не відкриваючи прямий доступ до даних. Методи класу, що описані всередині тіла класу, можна реалізувати поза класом за допомогою оператора дозволу області видимості `::`. Методи мають доступ до всіх полів класу через неявний

покажчик `this`, який вказує на об’єкт, для якого викликана функція. Це дозволяє уникнути конфліктів імен та забезпечує коректне звернення до елементів конкретного об’єкта.

Для створення об’єкта класу використовують синтаксис:

```
Name name;  // name – змінна типу Name, екземпляр класу
```

Після створення об’єкта можна звертатися до його полів і методів через оператор `.`:

```
name.field1;           // доступ до поля
name.method1(par1);    // виклик методу
```

Всі поля класу доступні з методів цього ж класу без передачі їх як параметрів. Рекомендується мінімізувати кількість відкритих полів і методів, адже прямий доступ до даних може призвести до помилок у роботі взаємопов’язаних методів. Загальне правило: чим менше публічних членів, тим безпечніше та стабільніше код.

Для ініціалізації об’єкта класу використовують конструктор. Ім’я конструктора збігається з ім’ям класу. Конструктор виконується автоматично при створенні об’єкта і використовується для присвоєння початкових значень полям класу. Конструктори можуть мати параметри, значення за замовчуванням, а також можуть бути перевантажені. Існує спеціальний конструктор копіювання, який дозволяє створити новий об’єкт на основі існуючого екземпляра класу:

```
Name(const Name &obj);
```

Конструктор копіювання отримує об’єкт класу за посиланням і створює копію його полів у новому об’єкті.

Для очищення ресурсів та завершення життєвого циклу об’єкта використовується деструктор. Деструктор виконується автоматично при виході об’єкта з області видимості або при видаленні з динамічної пам’яті. Його ім’я складається з символу ~ та імені класу. Деструктор не має параметрів і не може бути перевантаженим. Якщо деструктор не визначено явно, використовується стандартний деструктор компілятора.

Статичні елементи класу. Статичні функції-елементи не отримують вказівника `this` і тому не мають доступу до нестатичних елементів конкретного об'єкта. Вони застосовуються для роботи зі статичними даними класу або для виконання дій, що не залежать від конкретного об’єкта. Наприклад:

```
class Country
{
```

```
private:
    static char className[20];
public:
    static const char* getClassName() { return className; }
    static void setClassName(const char* name) { strcpy(className,
name); }
};
```

Виклик таких функцій можна здійснювати як через ім’я класу, так і через об’єкт:

```
Country::setClassName("Country"); // через ім’я класу
Country c;
c.setClassName("Contrée");        // через об’єкт
std::cout << Country::getClassName();
```

Як видно, будь-які зміни, здійснені через статичний елемент, будуть спільними для всіх об’єктів класу.

Наслідування. Об’єктно-орієнтований підхід передбачає можливість створення нових класів на основі вже існуючих. Цей механізм називається наслідуванням. Основні поняття:

- Базовий (батьківський) клас — клас, від якого наслідують.
- Похідний (дочірній) клас — клас, що успадковує властивості та методи базового класу.

Приклад:

```
class Country
{
protected:
    char name[40];
public:
    void setName(const char* n) { strcpy(name, n); }
    const char* getName() const { return name; }
};

class EuropeanCountry : public Country
{
private:
    bool inEU;
public:
    void setInEU(bool value) { inEU = value; }
    bool isInEU() const { return inEU; }
};
```

Ключове слово `public` визначає рівень доступу до членів базового класу у похідному класі. Можливі також варіанти `protected` та `private`.

Поліморфізм. Поліморфізм дозволяє об’єктам похідних класів

використовуватися через посилання або вказівники базового класу. Для реалізації поліморфізму у C++ застосовують віртуальні функції:

```
class Country
{
public:
    virtual void displayInfo() const
    {
        std::cout << "This is a country." << std::endl;
    }
};

class EuropeanCountry : public Country
{
public:
    void displayInfo() const override
    {
        std::cout << "This is a European country." << std::endl;
    }
};

void printCountryInfo(const Country& c)
{
    c.displayInfo(); // викликається відповідна реалізація
}
```

Виклик `printCountryInfo(EuropeanCountry())` виведе "This is a European country.", демонструючи роботу поліморфізму.

Абстракція та інтерфейси. Абстракція дозволяє виділяти лише необхідні властивості об’єкта, приховуючи деталі реалізації. У C++ для цього використовують чисто віртуальні функції, які оголошуються як = 0:

```
class ICountry
{
public:
    virtual double density() const = 0; // чисто віртуальна функція
    virtual ~ICountry() {}              // віртуальний деструктор
};
```

Клас, що наслідує `ICountry`, повинен реалізувати функцію `density()`.

Переваги об’єктно-орієнтованого підходу. Інкапсуляція — приховування внутрішніх деталей реалізації, забезпечення контролю доступу.

- Наслідування — повторне використання коду та розширення функціональності без дублювання.

- Поліморфізм — можливість взаємодії з різними типами об’єктів через єдиний інтерфейс.
- Абстракція — відокремлення суттєвих властивостей об’єкта від неважливих деталей.
- Модульність та структурованість коду — підвищення надійності та зрозумілості програми.

Об’єктно-орієнтований підхід дозволяє створювати програми, що максимально наближені до структури реальних систем, і полегшує масштабування складних проектів.

Обробка винятків. Дуже часто функція програми, у якій виникає певна помилка, не має можливості виправити її самостійно, бо невідомий контекст виклику цієї функції. Тому помилку необхідно передати до тієї частини програми, де її можна обробити.

- Вимоги до механізму сповіщення про помилки:
- Виключити можливість ігнорування помилки.
- Надати програмісту можливість гнучко реагувати на помилку.

Традиційні засоби програмування:

- Термінова зупинка програми – не завжди доцільна, якщо користувач може повторити введення або помилку можна виправити автоматично.
- Повернення коду помилки – код може бути випадково проігнорований, що небезпечно для програми.
- Зміна значення глобальної змінної – легко пропустити, потребує чисельних перевірок.

Винятки. Виняток – це подія, що порушує нормальне виконання програми під час виконання. Механізм генерації та обробки винятків дозволяє передати інформацію про помилку у місце, де її можна обробити.

Генерація винятку:

```
double reciprocal(double value)
{
    if (value == 0) {
        throw "Division by Zero"; // Генерація винятку типу char*
    }
    return 1 / value;
}
```

`throw` схожий на `return`, але повертає об’єкт винятку.

Тип винятку не пов’язаний з типом, який повертає функція.

Ключове слово `noexcept` (C++11+)

```
void g(int x) noexcept(true) { }
```

Використовується для функцій, що не генерують винятків.

Обробка винятків

```
try {
    double x = 0;
    cout << reciprocal(x);
}
catch (char* c) {
    cout << c; // "Division by Zero"
}
catch (...) {
    // Обробка всіх інших винятків
}
```

`catch` приймає тип винятку.

`catch(...)` ловить усі винятки.

Для повторної передачі винятку: `throw;`.

Використання власних типів винятків:

```
class MyClass {
public:
    class Bad_Data {
        int bad_value;
    public:
        Bad_Data(int value) : bad_value(value) {}
        int getBadValue() const { return bad_value; }
    };
    void f(int i) {
        if (i < 0) throw Bad_Data(i);
    }
};

int main() {
    try {
        MyClass m;
        m.f(-2);
    } catch (MyClass::Bad_Data& b) {
        cout << "Bad value: " << b.getBadValue();
    }
}
```

Рекомендовано використовувати об’єктні типи винятків.

У блоках `catch` об’єкти передаються за посиланням для ефективності.

Композиція класів. Об’єкти класів можна робити елементами

даних інших класів – це композиція класів. Конструктори внутрішніх об'єктів викликаються перед конструктором зовнішнього класу. Виклик конструкторів без параметрів відбувається автоматично. Деструктори викликаються у зворотному порядку.

Виклик конструкторів з параметрами:

```
class Part {
private:
    int someValue;
public:
    Part(int someValue) { this->someValue = someValue; }
};

class Whole {
private:
    Part part;
public:
    Whole(int someValue) : part(someValue) { }
};
```

Таким чином, класи у C++ дозволяють об'єднувати дані та функції в єдиний логічний блок, контролювати доступ до полів та методів за допомогою інкапсуляції, а також автоматично керувати ініціалізацією та очищенням ресурсів через конструктори і деструктори. Ці механізми є основою об'єктно-орієнтованого програмування, забезпечуючи надійність, повторне використання коду та логічну структурованість програм.

3.2. Приклади [5]

Приклад 3.1

Розглянемо клас `complex` для роботи з комплексними числами. У класі `complex` будуть члени класу: `double x` – дійсна частина комплексного числа, `double y` – уявна частина комплексного сила; методи класу: `double modul()` – функція обчислення модуля комплексного числа, `double argument()` – функція обчислення аргументу комплексного числа, `void show_complex()` – функція виводить комплексне число на екран.

```
#include <iostream>
#include <math.h>
#define PI 3.14159
using namespace std;
class complex
{
     public:
```

```
      double x;
      double y;
      double modul() {return pow (x * x + y * y, 0.5);}
      double argument() {return atan2 (y, x) * 180 / PI;}
      void show_complex () { if (y>= 0)
      cout << x << "+" << y << "i" << endl;
      else
      cout << x << y << "i" << endl;
}
};
int main ()
{
      complex chislo;
      chislo.x = 3.5;
      chislo.y = -1.432;
      chislo.show_complex();
      cout << "Modul ‘chisla =" << chislo.modul ();
      cout << endl << "Argument chisla =" << chislo.argument () << endl;
}
```

Приклад 3.2

Змінимо розглянутий раніше приклад класу `complex`. Додамо метод `vvod`, призначений для введення дійсної та уявної частини числа, члени класу і метод `show_complex` зробимо закритими, а інші методи відкритими. Текст програми буде мати вигляд:

```
#include <iostream>
#include <math.h>
#define PI 3.14159
using namespace std;
class complex
{
   public:
      void vvod();
      double modul();
      double argument();
      void show_complex();
   private:
      double x;
      double y;
};
void complex :: vvod ()
{
   cout << "Vvedite x \ t";
   cin >> x;
   cout << "Vvedite y \ t";
   cin >> y;
   show_complex ();
}
```

```
double complex :: argument()
{
   return atan2(y, x) * 180 / PI;
}
void complex :: show_complex()
{
   if (y>= 0)
        cout << x << "+" << y << "i" << endl;
   else cout << x << y << "i" << endl;
}
int main ()
{
complex chislo;
chislo.vvod();
cout<< "Modul kompleksnogo chisla =" << chislo.modul() << endl;
cout<<"Argument kompleksnogo chisla =" << chislo.argument();
}
```

У розглянутому прикладі показано спільне використання відкритих і закритих елементів класу. Поділ на відкриті і закриті в цьому прикладі кілька штучне, воно проведено лише для ілюстрації механізму спільного використання закритих або відкритих елементів класу. Якщо спробувати звернутися до методу `show_complex()` або до членам класу `x`, у з функції `main`, то компілятор видасть повідомлення про помилку (доступ до елементів класу заборонений).

Приклад 3.3

Додамо в створений в попередньому прикладі клас complex конструктор:

```
#include <iostream>
#include <math.h>
#define PI 3.14159
using namespace std;
class complex
{
   public:
      complex();
      double modul();
      double argument();
      void show_complex();

   private:
      double x;
      double y;
};
int main()
{
complex chislo;
cout<<"Modul kompleksnogo chisla ="<<chislo.modul()<<endl;
```

```
cout<<"Argument kompleksnogo chisla ="<<chislo.argument();
}
complex :: complex ()
{
    cout << "Vvedite x \t";
    cin >> x;
    cout << "Vvedite y \t";
    cin >> y;
    show_complex ();
}
double complex :: modul ()
{
    return pow (x * x + y * y, 0.5);
}
double complex :: argument ()
{
    return atan2 (y, x) * 180 / PI;
}
void complex :: show_complex ()
{
    if (y>= 0) cout << x << "+" << y << "i" << endl;
    else cout << x << y << "i" << endl;
}
```

3.2. Задачі

Реалізувати своє завдання, як багатомодульну програму

- h-файл, з об'явою класів;
- файл з реалізацією класів;
- головна програма.

Варіант 1

Створити клас квадрат, члени класу – довжина сторони. Передбачити у класі методи обчислення та виведення відомостей про фігуру – діагональ, периметр, площу.

Варіант 2

Створити клас трикутник, члени класу – довжини 3-х сторін. Передбачити у класі методи перевірки існування трикутника, обчислення та виведення відомостей про фігуру – довжини сторін, кути, периметр, площу.

Варіант 3

Створити клас коло, член класу – R. Передбачити у класі методи обчислення та виведення відомостей про фігуру – площу, довжину кола.

Варіант 4

Створити клас квадрат, члени класу – довжина сторони.

Передбачити у класі методи обчислення та виведення відомостей про фігуру – діагоналей, периметр, площу.

Варіант 5

Створити клас чотирикутник, члени класу – координати 4-х точок. Передбачити у класі методи перевірки існування чотирикутника обчислення та виведення відомостей про фігуру – довжини сторін, діагоналей, периметр, площу.

Варіант 6

Створити клас трикутник, члени класу – координати 3-х точок. Передбачити у класі методи перевірки існування трикутника, обчислення та виведення відомостей про фігуру – довжини сторін, кути, периметр, площу.

Варіант 7

Створити клас прямокутник, члени класу – довжини сторін a та b. Передбачити у класі методи обчислення та виведення відомостей про фігуру – довжини сторін, діагоналей, периметр, площу.

Варіант 8

Створити клас коло, член класу - R. Передбачити у класі методи обчислення та виведення відомостей про фігуру – площу, довжину кола.

Варіант 9

Створити клас чотирикутник, члени класу – координати 4-х точок. Передбачити у класі методи обчислення та виведення відомостей про фігуру – довжини сторін, діагоналей, периметр, площу.

Варіант 10

Створити клас рівносторонній трикутник, член класу – довжина сторони. Передбачити у класі методи обчислення та виведення відомостей про фігуру – периметр, площу.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ 4
Робота з файлами

Мета – опанувати написання програм з використанням файлів.

4.1. Теоретичні основи [6]

4.1.1 Робота з потоками символів у Java

Потоки, призначені для роботи з текстовою інформацією, мають назву потоків символів. Імена класів таких потоків закінчуються відповідно словами "...Reader" і "...Writer". Безпосередню роботу з текстовими файлами здійснюють об'єкти класів `FileReader` та `FileWriter`.

Важливий елемент роботи з файловими потоками – це буферизація. Буферизація передбачає створення в оперативній пам'яті спеціальної області (буферу), у яку дані завантажуються з файлу для подальшого поелементного читання або поелементно записуються дані з подальшим переписуванням на диск. Об'єкти класу `BufferedReader` здійснюють таке буферизоване читання.

Для буферизованого виведення застосовують об'єкти класу `BufferedWriter`. Безпосереднє форматоване виведення здійснюється методами `print()` та `println()` об'єкту класу `PrintWriter`.

У наведеному нижче прикладі з файлу з ім'ям data.txt здійснюється читання одного цілого й одного дійсного значення, їхня сума записується у файл `results.txt`.

```
import java.io.*;
import java.util.StringTokenizer;

public class FileTest {

    void readWrite() {
          FileReader fr = new FileReader("data.txt");
          BufferedReader br = new BufferedReader(fr);
          String s = br.readLine();
          int x;
          double y;
          StringTokenizer st = new StringTokenizer(s);
          x = Integer.parseInt(st.nextToken());
          y = Double.parseDouble(st.nextToken());
          double z = x + y;
          FileWriter fw = new FileWriter("results.txt");
          PrintWriter pw = new PrintWriter(fw);
```

```
            pw.println(z);
            pw.close();
    }

    public static void main(String[] args) {
        new FileTest().readWrite();
    }
}
```

Для відкриття файлу створюється об'єкт класу `FileReader`, у конструкторі якого вказується рядок – ім'я файлу. Посилання на створений об'єкт передається у конструктор класу `BufferedReader`. Читання з файлу здійснюється за допомогою методу `readLine()`, який повертає посилання на рядок символів, або `null`, якщо досягнуто кінець файлу.

Змінна s типу `String` посилається на рядок, який містить два числа. Для виділення з цього рядку окремих лексем використовують об'єкт класу `StringTokenizer`, у конструктор якого передається рядок. Посилання на окремі частини рядка поступово отримують за допомогою методу `nextToken()`. Ці посилання можуть бути використані безпосередньо, або використовуються для перетворення даних у числові значення (статичні методи `parseDouble()` та `parseInt()` класів `Double` та `Integer` відповідно).

Для читання з файлу можна використовувати клас `Scanner`. Фактичним параметром конструктора може бути файловий потік. Попередній приклад можна реалізувати за допомогою класу `Scanner`.

```
import java.io.*;
import java.util.Scanner;

public class FileTest {

    void readWrite() {
        Scanner scanner = new Scanner(new FileReader("data.txt"));
        PrintWriter pw = new PrintWriter("results.txt");
        pw.println(scanner.nextInt() + scanner.nextDouble());
    }

    public static void main(String[] args) {
        new FileTest().readWrite();
    }
}
```

4.1.2 Робота з текстовими файлами на C#

Як практично всі універсальні мови програмування C# надає засоби роботи з файлами та іншими потоками. Ці засоби описані у просторі імен `System.IO`. Класи цього простору імен пропонують низку методів для створення таких потоків, читання, запису тощо. Потоки, призначені для роботи з текстовою інформацією, мають назву потоків символів. Базовими класами для роботи з потоками символів є `TextReader` та `TextWriter`. Похідні класи `StreamWriter` та `StreamReader`, а також похідні від них, забезпечують роботу з текстовими файлами.

Наведена нижче програма здійснює читання з текстового файлу рядків та запис їх в інший текстовий файл.

```
using System;
using System.IO;

namespace LabThird
{
    class Program
    {
        static void Main(string[] args)
        {
            using (StreamReader reader = new StreamReader("From.txt",
Encoding.Default))
            {
                using (StreamWriter writer = new StreamWriter("To.txt"))
                {
                    string s;
                    while ((s = reader.ReadLine()) != null)
                    {
                        writer.WriteLine(s);
                    }
                }
            }
        }
    }
}
```

Завдяки вживанню оператора `using()` файли автоматично закриваються, оскільки викликаються методи `Dispose()`, які своєю чергою викликають методи `Close()`. Файл `From.txt` повинен до початку роботи програми знаходитися у теці `bin\Debug` або `bin\Release` проєкту (залежно від способу завантаження програми на виконання).

Існує функція `ReadToEnd()`, яка дозволяє прочитати весь файл до кінця та занести весь вміст в один рядок.

Для роботи з двійковими потоками використовують класи `BinaryReader` і `BinaryWriter`. Також існують так звані потоки в пам'яті – `StringReader` і `StringWriter`, які дозволяють використовувати рядки як потоки введення та виведення.

4.2. Приклади [6]

Приклад 4.1

Порядкове копіювання текстових файлів на Java [6].

```
import java.io.*;

public class TextFileCopy {

    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Необхідні аргументи!");
            return;
        }
        try (BufferedReader in = new BufferedReader(new
FileReader(args[0]));
             PrintWriter out = new PrintWriter(new
FileWriter(args[1]))) {
            String line;
            while ((line = in.readLine()) != null) {
                out.println(line);
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Приклад 4.2

Порядкове копіювання текстових файлів на C# [6].

```
using System.IO;

namespace LabThird
{
    class Program
    {
        static void Main(string[] args)
        {
            using (StreamReader reader = new StreamReader("From.txt",
Encoding.Default))
```

```
            {
                using (StreamWriter writer = new
StreamWriter("To.txt"))
                {
                    string s = reader.ReadToEnd();
                    writer.Write(s);
                }
            }
        }
    }
}
```

4.3. Задачі

Варіант 1

Реалізувати програму читання з текстового файлу цілих додатних значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих чисел у масив, сортування за зменшенням та за збільшенням суми цифр та зберігання обох результатів у двох нових текстових файлах.

Варіант 2

Реалізувати програму читання з текстового файлу цілих додатних значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих чисел у масив, сортування за зменшенням та за збільшенням та зберігання обох результатів у двох нових текстових файлах.

Варіант 3

Реалізувати програму читання з текстового файлу дійсних додатних значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих чисел у масив, сортування за зменшенням та за збільшенням та зберігання обох результатів у двох нових текстових файлах.

Варіант 4

Реалізувати програму читання з текстового файлу цілих значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих чисел у масив, сортування за зменшенням та за збільшенням та зберігання обох результатів у двох нових текстових файлах.

Варіант 5

Реалізувати програму читання з текстового файлу дійсних значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих

чисел у масив, сортування за зменшенням та за збільшенням та зберігання обох результатів у двох нових текстових файлах.

Варіант 6

Реалізувати програму читання з текстового файлу цілих додатних значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих чисел у масив, сортування за зменшенням та за збільшенням суми цифр та зберігання обох результатів у двох нових текстових файлах.

Варіант 7

Реалізувати програму читання з текстового файлу цілих додатних значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих чисел у масив, сортування за зменшенням та за збільшенням та зберігання обох результатів у двох нових текстових файлах.

Варіант 8

Реалізувати програму читання з текстового файлу дійсних додатних значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих чисел у масив, сортування за зменшенням та за збільшенням та зберігання обох результатів у двох нових текстових файлах.

Варіант 9

Реалізувати програму читання з текстового файлу цілих значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих чисел у масив, сортування за зменшенням та за збільшенням та зберігання обох результатів у двох нових текстових файлах.

Варіант 10

Реалізувати програму читання з текстового файлу дійсних значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих чисел у масив, сортування за зменшенням та за збільшенням та зберігання обох результатів у двох нових текстових файлах.

ВИМОГИ ДО ЗВІТУ З ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Виконаний звіт з індивідуального завдання може бути подано у вигляді текстового файлу у форматі *.doc (.docx)* і прикріплено до відповідної активності в системі Moodle у форматі *.pdf*.

Звіт виконується українською мовою з дотриманням норм наукового стилю, який передбачає:

- формально-логічний спосіб подання матеріалу, аргументовані міркування, що сприяють доведенню істинності положень і обґрунтуванню основних висновків дослідження;
- змістову завершеність, цілісність та логічну зв'язність викладу;
- об'єктивність, цілеспрямованість і відсутність емоційного забарвлення тексту.

Структура звіту:

- титульний аркуш (Додаток А);
- постанова задачі;
- опис програми (класів, методів);
- керівництво користувача (скріни працюючої програми);
- висновки.
- додаток (код програми).

Робота подається у вигляді текстового файлу текст роботи повинен бути виконаний у вигляді комп'ютерного набору на одному боці аркуша білого паперу формату А4 (210х297мм). Текст кожного індивідуального завдання здобувача розміщується на аркуші книжкової або альбомної орієнтації, яка обмежується полями: лівим – 30 мм, правим – 10 мм, верхнім – 20 мм, нижнім – 20 мм. Для великих таблиць і рисунків допускається альбомна орієнтація сторінок, на яких вони розміщені. Текст роботи оформлюється шрифтом Arial, кеглем 14 з одинарним міжрядковим інтервалом. Для таблиць допускається використання шрифту Arial, кеглем 12.

Заголовки пунктів у разі їх виділення слід починати з абзацного відступу і друкувати маленькими літерами, крім першої великої, не підкреслюючи, без крапки в кінці. Абзацний відступ повинен бути однаковим упродовж усього тексту і дорівнювати 1,25 см. Якщо заголовок складається з двох і більше речень, їх розділяють крапкою. Перенесення слів у заголовку не допускається.

Відстань між заголовком і попереднім текстом повинна бути два рядки, між заголовком і подальшим текстом – один рядок.

Не допускається розміщувати назву пункту в нижній частині сторінки, якщо після неї розміщено тільки один рядок тексту.

Сторінки роботи нумеруються арабськими цифрами з наскрізною нумерацією по всьому тексту. Номер сторінки розміщується у правому верхньому куті без крапки в кінці. Титульний аркуш та зміст входять до

загальної нумерації, але номер сторінки на них не проставляється. Ілюстрації та таблиці, які подані на окремих сторінках, також включаються до загальної нумерації. Пункти роботи нумеруються арабськими цифрами без крапки після номера.

Ілюстрації (рисунки, графіки, схеми, діаграми) повинні розміщуватися одразу після тексту, де вони згадуються вперше, або на наступній сторінці. У тексті роботи мають бути обов’язкові посилання на всі ілюстрації.

Цифрові дані зазвичай подаються у вигляді таблиць. Таблиці розміщуються безпосередньо після тексту, де вони вперше згадуються, або на наступній сторінці. У тексті повинні бути відповідні посилання на всі таблиці.

Назва таблиці складається зі слова «Таблиця», її порядкового номера та заголовка, який стисло відображає зміст поданих у ній даних. Повна назва таблиці зазначається один раз над таблицею зліва, з абзацним відступом.

Якщо таблиця переноситься на наступну сторінку, над її продовженням із абзацного відступу пишуть: «Продовження таблиці Х» або «Кінець таблиці Х», де Х – номер таблиці. Таблиці нумеруються арабськими цифрами послідовно в межах усієї роботи.

Заголовки та дані таблиці можуть бути оформлені через одинарний інтервал, шрифтом Arial, 12 кегль. Заголовки граф починають із великої літери, а підзаголовки – з малої, якщо вони становлять одне речення із заголовком. Якщо підзаголовки мають самостійне значення, їх пишуть з великої літери. У кінці заголовків і підзаголовків крапка не ставиться. Усі заголовки та підзаголовки граф подаються в однині.

ПОДАННЯ НА ПЕРЕВІРКУ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ ТА КРИТЕРІЇ ОЦІНЮВАННЯ

Критерії оцінювання результатів виконання практичної роботи.

Захист індивідуального завдання відбувається на практичному занятті згідно з графіком контрольних точок, передбаченим робочою програмою дисципліни, а оцінка за його виконання виставляється викладачем у відповідній активності в системі Moodl і враховується ним при визначенні поточної успішності здобувача.

Максимальна кількість балів, яку здобувач може отримати за кожне виконане індивідуальне завдання – 15 балів. Оскарження оцінки може бути здійснене на останньому практичному занятті модуля.

Критерії оцінювання:

- здобувач вищої освіти підготував звіт з індивідуального завдання за своїм варіантом у вигляді файлу *pdf, в якому надав: умову задачі, скріни коду програми та її виконання у програмному забезпеченні, додаток з кодом програми (10 балів);
- здобувач вищої освіти захистив індивідуальне завдання: продемонстрував роботу програми та відповів на запитання викладача (5 балів).

Додаткові зауваження:

- здобувач вищої освіти може оскаржити отримані оцінки в порядку, передбаченому Положенням про організацію освітнього процесу та Положенням про політику та процедури врегулювання конфліктних ситуацій;
- викладач не має права знижувати оцінку за індивідуальне завдання, якщо воно не було складено вчасно, однак в разі, якщо така робота була оцінена пізніше.

СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1 Порєв В. М. Об'єктно-орієнтоване програмування : конспект лекцій. Київ : КПІ ім. Ігоря Сікорського, 2022. 271 с.

2 Malhotra D., Malhotra N. C++ Programming Fundamentals. Mercury Learning and Information, 2023. 351 p.

3 McClanahan P. C++ Programming I. LibreTexts, 2021. URL: https://read.kortext.com/reader/pdf/996893

4 McClanahan P. C++ Data Structures. LibreTexts, 2021. URL: https://read.kortext.com/reader/pdf/996892

5 Методичні вказівки до виконання лабораторних робіт з об’єктно-орієнтоване програмування на С++ за освітньо-професійною програмою першого (бакалаврського) рівня спеціальності 122 «Комп’ютерні науки» / Уклад. Нікуліна О.М. Запоріжжя, ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», 2024. 52 с. URL: https://dspace.mipolytech.education/items/7509255e-ed80-4a0a-9e20-ab35cf38e1d6

Web-ресурси

6 Іванов Л. В. Навчальні курси: Iwanoff. URL: http://www.iwanoff.inf.ua/ (дата звернення: 20.10.2025).

7 C# Підручник : W3SchoolsUA.. URL: https://w3schoolsua.github.io/cs/index.html#gsc.tab=0 (дата звернення: 20.10.2025).

8 C++ Підручник : W3SchoolsUA. URL: https://w3schoolsua.github.io/cpp/index.html#gsc.tab=0 (дата звернення: 20.10.2025).http://www.iwanoff.inf.ua/programming_2_ua/index.html

.

Додаток А

ПРИКЛАД ОФОРМЛЕННЯ ТИТУЛЬНОГО ЛИСТА ПРАКТИЧНОЇ РОБОТИ

ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»
Кафедра цифрових технологій та проєктно-аналітичних рішень

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ № 1

з навчальної дисципліни
«АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ»

РОБОТА ЗІ СТАТИЧНІ МАСИВИ НА МОВІ С++

Виконав (ла): здобувач (ка) вищої освіти
за освітньо-професійною програмою
«Комп'ютерні науки»
гр. 0122-хх-х

_____
(Прізвище, ім'я, по батькові повністю)

Прийняв:

Запоріжжя – 202_

Навчально-методичне видання

Олена Миколаївна Нікуліна
Олексій Михайлович Кондратов

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ

методичні рекомендації
до виконання індивідуальних завдань

самостійне електронне мережеве видання

Публікується в авторській редакції