МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра «Програмна інженерія та   
інформаційні технології управління»

ЗВІТ

з лабораторної роботи № 2

з дисципліни

«ОСНОВИ ПРОГРАМУВАННЯ»

Виконала

Студентка групи КН-222В

Григор’єва Катерина Ігорівна

Перевірив

Асистент каф. ПІІТУ

Олексій КОНДРАТОВ

Харків 2023

**Тема:** Створення та використання класів C++.

**Мета:** Навчитися створювати та використовувати класи у мові програмування C++, зрозуміти принципи ООП, розглянути основні концепції, такі як інкапсуляція, успадкування та поліморфізм. Також дослідити імплементацію конструкторів та деструкторів класів, розглянути патерни проектування та їхнє застосування у створенні класів. Після завершення курсу студенти повинні бути в змозі створювати класи у C++ з урахуванням принципів ООП та розуміти, як вони можуть бути використані для побудови складних програмних систем.

## **1 Завдання на лабораторну роботу**

### **1.1 Клас для представлення точки в тривімірному просторі**

Створити клас для опису точки в тривимірному просторі. Клас повинен відповідати таким вимогам:

* мати конструктор без параметрів;
* мати конструктор с трьома параметрами;
* містити елементи даних типу **double** для представлення координат точки;
* реалізовувати публічні функції доступу до даних (сеттери й геттери).

Для обчислення відстані створити операторну функцію **operator**-() ("мінус") з двома параметрами типу класу для представлення точок і результатом типу **double**. Функцію оголосити як друга класу.

У функції main() слід створити два об'єкти типу точки в тривімірному просторі, застосувавши різні конструктори. Для встановлення і читання значень застосувати функції доступу. Обчислити відстань між двома точками через використання операції "мінус", потім змінити координати однієї з точок і обчислити відстань між двома точками через явний виклик функції **operator**-().

### **1.2 Клас для представлення простого дробу**

Створити клас для представлення простого дробу. Реалізувати конструктори, функцію скорочення дробу, а також перевантажити операції +, -, \*, /, <, <=, >, >=, введення та виведення. В операторній функції виведення реалізувати максимально коректне виведення: отримувати та виводити цілу частину для неправильних дробів, не виводити знаменник, якщо він дорівнює 1 або чисельник дорівнює 0 тощо.

Здійснити читання двох дробів та демонстрацію скорочення дробів і всіх перевантажених операцій в функції main().

Примітка: не слід окремо зберігати цілу частину дробу, оскільки вона завжди може бути розрахована з чисельника і знаменника.

### **1.3 Класи для представлення студента і групи**

Створити класи для представлення даних про студента і групи студентів. Клас "Студент" повинен містити такі приватні елементи даних:

* номер студентського посвідчення (**unsigned int**);
* прізвище (вказівник на символ); відповідний рядок створюватиметься у динамічній пам'яті за необхідністю;
* оцінки за останню сесію у вигляді масиву цілих від 0 до 100 (оцінки за предметами); масив оцінок створюється в динамічній пам'яті;
* вказівник на об'єкт класу "Група".

Визначити в класі "Студент" такі елементи:

* конструктор без параметрів і конструктор з параметрами;
* конструктор копіювання;
* деструктор;
* функції доступу до даних (сеттери й геттери);
* перевантажена операцію присвоєння.

Операторну функцію виведення даних про студента в потік слід реалізувати як зовнішню функцію – друга класу. Для забезпечення сортування перевантажити необхідні функції порівняння (відношення).

Слід визначити константу для максимально можливої кількості студентів (наприклад, 50)

У класі "Група" слід визначити такі елементи даних:

* індекс групи;
* масив указівників на студентів максимально можливої довжини;
* реальна кількість указівників у масиві.

Визначити в класі "Група" такі елементи:

* конструктор без параметрів і конструктор з параметрами;
* функції доступу до даних про індекс (сеттер і геттер);
* перевантажена операцію присвоєння;
* функцію сортування за визначеним критерієм;
* знаходження студентів за певною ознакою.

Замість геттера для масиву вказівників можна перевантажити операцію отримання елемента за індексом. Крім того, потрібна функція виведення в потік.

У функції main() створити об'єкт "Група", додати масив студентів і продемонструвати всі реалізовані функції, зокрема, сортування і пошук за ознакою.

### **1.4 Клас для представлення двовимірного масиву**

Розробити клас для представлення двовимірного масиву (матриці) цілих чисел довільних розмірів. Створити конструктори та деструктор, перевантажити операції додавання, віднімання і множення (згідно з правилами роботи з матрицями), звертання за індексом, введення з потоку та виведення в потік. Створити власні класи винятків та генерувати відповідні об'єкти-винятки, якщо неможливо виконати ту чи іншу операцію.

Створити окрему функцію, яка отримує посилання на матрицю і виконує над масивом дії, вказані в таблиці. Функція не повинна бути методом класу або дружньою функцією.



У функції main() здійснити тестування всіх можливостей класу з перехопленням можливих винятків, а також розв'язати індивідуальну задачу.

### **1.5 Підрахунок суми введених значень**

Створити клас з одним закритим елементом даних цілого типу, геттером і конструктором з одним параметром. В цьому ж класі створити закрите статичне поле, яке зберігає суму цілих елементів даних всіх раніше створених об'єктів. Під час кожного виклику конструктора до статичного поля повинно додаватися нове значення. Статична публічна функція цього ж класу повинна повертати цю суму.

У функції main() створити декілька об'єктів і вивести отриману суму.

2 Завдання

Перше завдання

Код до першого завдання

#include <iostream>

#include <cmath>

// Клас для представлення точки на екрані

class Point3D

{

friend double operator-(const Point3D &p1, const Point3D &p2)

{

double dx = p1.x - p2.x;

double dy = p1.y - p2.y;

double dz = p1.z - p2.z;

return sqrt(dx \* dx + dy \* dy + dz \* dz);

}

private:

double x, y, z; // координати точки

public:

// Конструктор без параметрів викликає інший конструктор:

Point3D() : Point3D(0, 0, 0) {}

// Конструктор ініціалізує елементи даних:

Point3D(double x, double y, double z) : x(x), y(y), z(z) {}

double getX() const { return x; }

void setX(double x) { this->x = x; }

double getY() const { return y; }

void setY(double y) { this->y = y; }

double getZ() const { return z; }

void setZ(double z) { this->z = z; }

// Допоміжна статична функція обчислення другого степеня:

static double sqr(double x) { return x \* x; }

};

int main()

{

// Першу точку створюємо за допомогою конструктора з параметрами:

Point3D p1;

Point3D p2(1, 2, 3);

std::cout << p1.getX() << " " << p1.getY() << " " << p1.getZ() << "\n";

std::cout << p2.getX() << " " << p2.getY() << " " << p2.getZ() << "\n";

double distance = p2 - p1;

std::cout << "Distance between p1 and p2: " << distance << std::endl;

// Для другої точки координати вказуємо сеттерами:

p2.setX(4);

p2.setY(6);

p2.setZ(7);

std::cout << p2.getX() << " " << p2.getY() << " " << p2.getZ() << "\n";

// Обчислюємо відстань:

std::cout << "Distance between p1 and p2: " << operator-(p1, p2) << std::endl;

return 0;

}

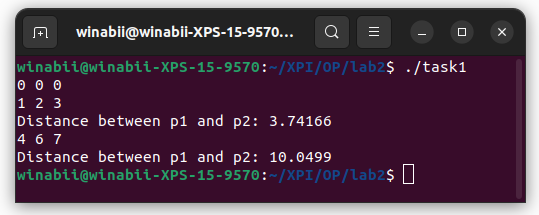


Рис 1 – Приклад роботи першої програми

Друге завдання

Код до другого завдання

#include <iostream>

using std::cerr;

using std::cin;

using std::cout;

using std::endl;

using std::invalid\_argument;

using std::istream;

using std::ostream;

#include <cmath>

int findGCD(int a, int b)

{ // алгоритм Евкліда

if (b == 0)

{

return a;

}

return findGCD(b, a % b);

}

class Fraction

{

// Введення та виведення

friend istream &operator>>(istream &in, Fraction &fraction)

{

if (fraction.denominator == 0)

{

throw std::invalid\_argument("Denominator cannot be zero!");

}

return in >> fraction.numerator >> fraction.denominator;

};

friend ostream &operator<<(ostream &out, const Fraction &fraction)

{

if (fraction.numerator == 0)

{

out << 0;

}

else if (fraction.denominator == 1)

{

out << fraction.numerator;

}

else if (abs(fraction.numerator) > abs(fraction.denominator))

{

int integerPart;

if (fraction.numerator == 0)

{

integerPart = 0;

}

else

{

integerPart = fraction.numerator / fraction.denominator;

}

int newNumerator = fraction.numerator - integerPart \* fraction.denominator;

if (integerPart != 0)

{

out << integerPart << " ";

}

out << newNumerator << "/" << fraction.denominator;

}

else

{

out << fraction.numerator << "/" << fraction.denominator;

}

return out;

}

// Перевантажені оператори

friend Fraction operator+(Fraction fraction1, Fraction fraction2)

{

double sumNumerator = fraction1.numerator \* fraction2.denominator + fraction1.denominator \* fraction2.numerator;

double sumDenominator = fraction1.denominator \* fraction2.denominator;

Fraction sum(sumNumerator, sumDenominator);

sum.simplify();

return sum;

};

friend Fraction operator-(Fraction fraction1, Fraction fraction2)

{

double subNumerator = fraction1.numerator \* fraction2.denominator - fraction1.denominator \* fraction2.numerator;

double subDenominator = fraction1.denominator \* fraction2.denominator;

Fraction sub(subNumerator, subDenominator);

sub.simplify();

return sub;

};

friend Fraction operator\*(Fraction fraction1, Fraction fraction2)

{

int mulNumerator = fraction1.numerator \* fraction2.numerator;

int mulDenominator = fraction1.denominator \* fraction2.denominator;

Fraction mul(mulNumerator, mulDenominator);

mul.simplify();

return mul;

};

friend Fraction operator/(Fraction fraction1, Fraction fraction2)

{

int divNumerator = fraction1.numerator \* fraction2.denominator;

int divDenominator = fraction1.denominator \* fraction2.numerator;

Fraction div(divNumerator, divDenominator);

div.simplify();

return div;

};

friend bool operator>=(const Fraction &fraction1, const Fraction &fraction2)

{

return fraction1.numerator \* fraction2.denominator >= fraction2.numerator \* fraction1.denominator;

}

friend bool operator<=(const Fraction &fraction1, const Fraction &fraction2)

{

return (fraction1.numerator \* fraction2.denominator <= fraction2.numerator \* fraction1.denominator);

}

friend bool operator>(const Fraction &fraction1, const Fraction &fraction2)

{

return (fraction1.numerator \* fraction2.denominator > fraction2.numerator \* fraction1.denominator);

}

friend bool operator<(const Fraction &fraction1, const Fraction &fraction2)

{

return (fraction1.numerator \* fraction2.denominator < fraction2.numerator \* fraction1.denominator);

}

private:

int numerator; // чисельник

int denominator; // знаменник

public:

// Конструктори

Fraction() { numerator = denominator = 1; };

Fraction(int numerator, int denominator);

// Функція скорочення дробу

void simplify()

{

int gcd = findGCD(numerator, denominator);

if (gcd != 0)

{

numerator /= gcd;

denominator /= gcd;

}

}

// Геттери:

double getNumerator() { return numerator; }

double getDenominator() { return denominator; }

};

// Конструктор з параметрами

Fraction::Fraction(int numerator, int denominator)

{

this->numerator = numerator;

this->denominator = denominator;

}

int main()

{

// Створюємо два об'єкти й читаємо з клавіатури їх координати:

Fraction fraction1, fraction2;

cout << "Enter the numerator and denominator with a space in the first fraction: ";

while (true)

{

try

{

cin >> fraction1;

break;

}

catch (const invalid\_argument &e)

{

cerr << e.what() << "\n"; // cerr - поток для виводу помилок

cerr << "Enter the numerator and denominator with a space in the first fraction: ";

}

}

cout << "Enter the numerator and denominator with a space in the second fraction: ";

try

{

cin >> fraction2;

if (fraction2.getDenominator() == 0)

{

throw invalid\_argument("Denominator cannot be zero!");

}

cout << "The second fraction is: " << fraction2 << endl;

}

catch (const invalid\_argument &e)

{

cerr << "Invalid input: " << e.what() << endl;

return 1;

}

// Демонструємо роботу перевантажених операцій:

cout << "Sum of numbers: " << fraction1 + fraction2 << endl;

cout << "Difference in numbers: " << fraction1 - fraction2 << endl;

cout << "Multiplication of numbers: " << fraction1 \* fraction2 << endl;

cout << "Division of numbers: " << fraction1 / fraction2 << endl;

cout << "Greater than: " << (fraction1 > fraction2 ? "true" : "false") << endl;

cout << "Greater than or equal to: " << (fraction1 >= fraction2 ? "true" : "false") << endl;

cout << "Less than: " << (fraction1 < fraction2 ? "true" : "false") << endl;

cout << "Less than or equal to: " << (fraction1 <= fraction2 ? "true" : "false") << endl;

return 0;

}

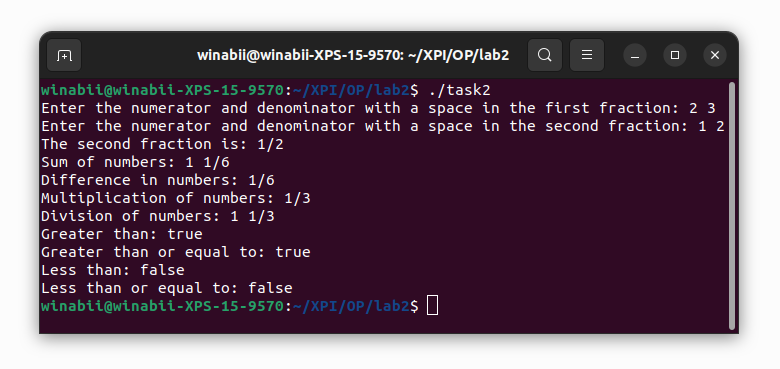


Рис 2 – Приклад роботи другої програми

Третє завдання

Код до третього завдання

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstring>

#include <iostream>

using std::cout;

using std::endl;

using std::strcpy;

using std::strlen;

const int MAX\_COUNT = 50; // Максимальна кількість студентів у группі

// Треба заздалегідь оголосити клас, щоб можна було створювати вказівник:

class Group;

// Клас для представлення міста

class Student

{

// Перевантажений оператор для виведення в потік

friend std::ostream &operator<<(std::ostream &out, const Student &student);

private:

unsigned int IDNumber; // Номер студентського посвідчення

char \*surname = nullptr; // прізвище, яке виділено у дінамічній пам'яті

int \*myArray;

int arraySize = 0;

Group \*group = nullptr; // вказівник на групу

public:

// Конструктори:

Student() {}

Student(unsigned int IDNumber, const char \*surname, int \*myArray, int arraySize, Group \*group);

Student(const Student &student);

~Student()

{

delete[] myArray; // звільняємо виділену пам'ять для масиву

}

void fillArray()

{

for (int i = 0; i < arraySize; i++)

{

myArray[i] = i + 1;

}

}

// Геттери:

unsigned int getIDNumber() const { return IDNumber; }

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

int \*getArray() const { return myArray; }

int getArraySize() const { return arraySize; }

Group \*getGroup() const { return group; }

// Сеттери:

void setIDNumber(unsigned int number) { IDNumber = number; }

void setSurname(const char \*surname)

{

if (this->surname != nullptr) // перевірка наявності попереднього значення прізвища

{

delete[] this->surname; // звільнення пам'яті, виділеної для попереднього значення

}

this->surname = new char[strlen(surname) + 1]; // виділення пам'яті для зберігання нового значення прізвища

strcpy(this->surname, surname); // копіювання значення прізвища у виділену пам'ять

}

void setMyArray(int \*array, int size)

{

arraySize = size;

myArray = new int[size];

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

{

myArray[i] = array[i];

}

}

void setGroup(Group \*group) { this->group = group; }

// Перевантажена операція присвоєння

const Student &operator=(const Student &student);

};

// Конструктор з параметрами, реалізований через виклик сеттерів

Student::Student(unsigned int IDNumber, const char \*surname, int \*myArray, int arraySize, Group \*group)

{

setIDNumber(IDNumber);

setSurname(surname);

setMyArray(myArray, arraySize);

setGroup(group);

}

// Конструктор копіювання

Student::Student(const Student &student)

{

IDNumber = student.IDNumber;

// Виділяємо пам'ять для нового рядка та копіюємо значення з оригінального рядка

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

strcpy(surname, student.surname);

// Виділяємо пам'ять для нового масиву та копіюємо значення з оригінального масиву

arraySize = student.arraySize;

myArray = new int[arraySize];

for (int i = 0; i < arraySize; i++)

{ // копіюємо значення з оригінального масиву

myArray[i] = student.myArray[i];

}

group = student.group;

// Присвоюємо вказівник на ту саму групу, що й у оригінальному об'єкті

IDNumber = student.IDNumber;

}

class Group

{

// Перевантажений оператор для виведення в потік

friend std::ostream &operator<<(std::ostream &out, const Group &group)

{

out << "Group Index: " << group.groupIndex << "\n\n";

for (int i = 0; i < group.count; i++)

{

out << \*(group.student[i]) << endl;

}

out << endl;

return out;

}

private:

Student \*student[MAX\_COUNT] = {}; // масив указівників на студентів

unsigned int groupIndex; // індекс групи

int count = 0; // кількість вказівників у масиві

public:

// Конструктори:

Group() {}

Group(unsigned int groupIndex) { setGroupIndex(groupIndex); }

// Геттер

const char getGroupIndex() const { return groupIndex; }

// Сеттери:

void setGroupIndex(unsigned int Index) { groupIndex = Index; }

void setGroups(Student \*student[], int count);

// Перевантажена операція присвоєння

const Group &operator=(const Group &group);

// Сортування за фамілією

void sortBySurname();

// Вивід за ознакою

void printIf();

// перевантаження операції отримання елемента за індексом

Student &operator[](int index)

{

return \*student[index];

}

};

// Отримаємо з параметру й заповнюємо масив студентів

void Group::setGroups(Student \*student[], int count)

{

this->count = count;

for (int i = 0; i < count; i++)

{

this->student[i] = student[i];

this->student[i]->setGroup(this); // this позволяет получить доступ к текущему объекту класса Group.

// Затем вызывается метод setGroup() [который находитя у объекта класса Student],

// передавая ему указатель на текущий объект класса Group, чтобы установить связь между объектами Group и Students

}

}

// Сортировка по фамилии

void Group::sortBySurname()

{

bool mustSort;

do

{

mustSort = false;

for (int i = 0; i < count - 1; i++)

{

if (strcmp(student[i]->getSurname(), student[i + 1]->getSurname()) > 0)

{

Student \*temp = student[i];

student[i] = student[i + 1];

student[i + 1] = temp;

mustSort = true;

}

}

} while (mustSort);

}

std::ostream &operator<<(std::ostream &out, const Student &student)

{

char buffer[300];

sprintf(buffer, "ID: %u\nSurname: %s\nArray: \n", student.IDNumber, student.surname);

for (int i = 0; i < student.arraySize; i++)

{

sprintf(buffer + strlen(buffer), "%d ", student.myArray[i]);

// buffer + strlen(buffer)получние указателя на позицию в буфере, которая находится после последнего записанного символа

//(с помощью strlen(buffer) мы получаем длину строки, которую уже записали в буфер, и к указателю buffer прибавляем эту длину).

// buffer указывает на первый элемент массива

}

sprintf(buffer + strlen(buffer), "\n");

out << buffer;

return out;

}

void Group::printIf()

{

for (int i = 0; i < count; i++)

{

if ((student[i]->getIDNumber()) % 2 != 0)

{

cout << \*student[i];

}

}

}

int main()

{

setlocale(LC\_ALL, "UKRAINIAN");

const int realCount = 4; // працюємо з чотирма студентами

Student \*student[realCount]; // створюємо масив вказівників на студентів

// заповнюємо масив

int tempArray[] = {100, 100, 100, 100, 100, 100, 100, 100};

student[0] = new Student(1421125, "Іванів", tempArray, 8, nullptr);

student[1] = new Student(284942, "Кучма", tempArray, 8, nullptr);

student[2] = new Student(54618, "Константінов", tempArray, 8, nullptr);

student[3] = new Student(264753, "Григор'єва", tempArray, 8, nullptr);

Group group = 211432;

group.setGroups(student, realCount);

cout << "Display the group data: \n"

<< group << "\n\n"; // виводимо всі дані

cout << "Display information about the student by index: \n"

<< \*student[0] << "\n\n";

group.sortBySurname();

cout << "Display all data after sorting\n"

<< group << "\n\n";

cout << "Remove students with odd student ID numbers:"

<< "\n\n";

group.printIf();

// Видалення студентів, на які вказують вказівники в масиві student

for (int i = 0; i < realCount; i++)

{

delete student[i];

}

return 0;

}

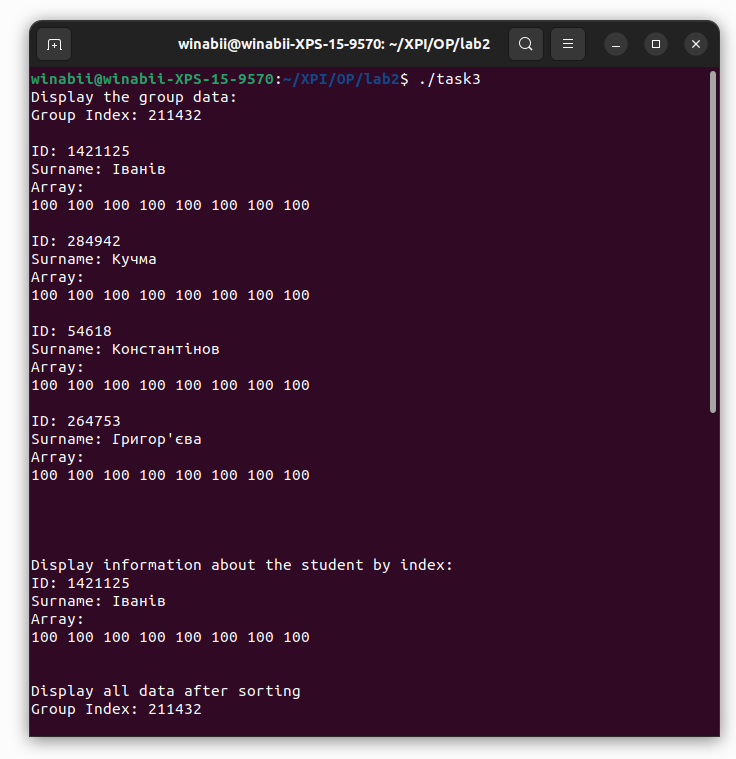


Рис 3 – Приклад роботи третьої програми

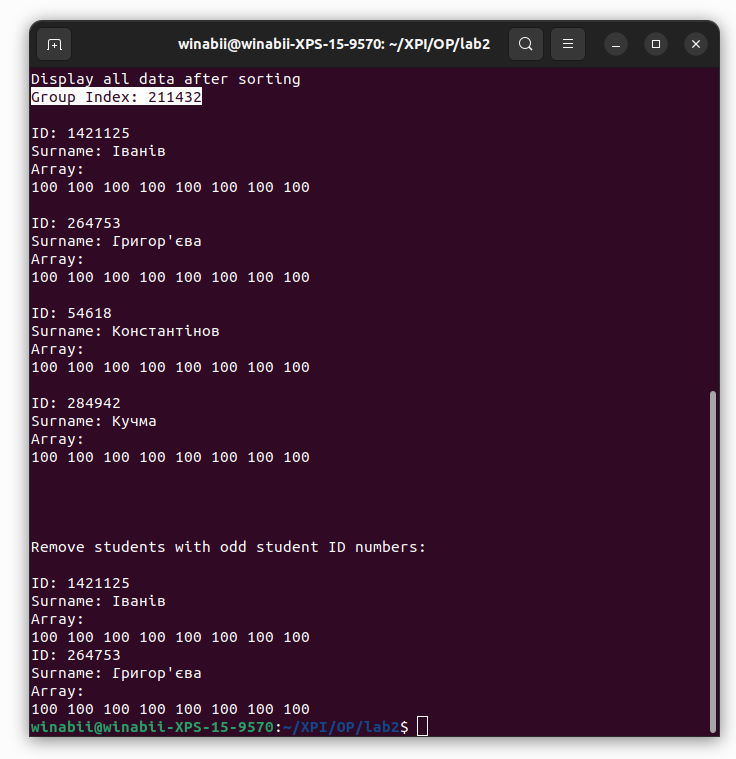


Рис 3.1 – Приклад роботи третьої програми

Четверте завдання

Код до четвертого завдання

#include <iostream>

#include <math.h>

using std::cin;

using std::cout;

using std::endl;

using std::istream;

using std::ostream;

// Клас для представлення 2го масиву

class UnsignedIntArray

{

// Дружні функції перевантаження операцій виведення та введення:

friend ostream &operator<<(ostream &out, const UnsignedIntArray &a);

friend istream &operator>>(istream &in, UnsignedIntArray &a);

friend UnsignedIntArray operator+(const UnsignedIntArray &a, const UnsignedIntArray &b);

friend UnsignedIntArray operator-(const UnsignedIntArray &a, const UnsignedIntArray &b);

friend UnsignedIntArray operator\*(const UnsignedIntArray &a, const UnsignedIntArray &b);

private:

int \*\*pa; // вказівник на майбутній масив

int rows = 0; // поточний розмір масиву

int cols = 0; // розмір другого розміру масиву (кількість стовпців).

public:

// Вкладений клас для створення об'єкту-винятку

class OutOfBounds

{

int index; // індекс за межами діапазону

public:

OutOfBounds(int i) : index(i) {} // конструктор

int getIndex() const { return index; } // геттер для індексу

};

// Конструктори:

UnsignedIntArray() {}

UnsignedIntArray(int r, int c)

{

if (r <= 0)

{

throw OutOfBounds(r);

}

if (c <= 0)

{

throw OutOfBounds(c);

}

rows = r;

cols = c;

pa = new int \*[rows];

for (int i = 0; i < rows; i++)

{

pa[i] = new int[cols];

}

}

UnsignedIntArray(const UnsignedIntArray &arr);

// Деструктор

~UnsignedIntArray()

{

for (int i = 0; i < rows; i++)

{

delete[] pa[i];

}

delete[] pa;

}

int \*operator[](int index)

{

if (index < 0 || index >= rows)

{

throw OutOfBounds(index);

}

return pa[index];

}

// Геттери

int getRows() const { return rows; };

int getCols() const { return cols; };

};

// Конструктор копіювання

UnsignedIntArray::UnsignedIntArray(const UnsignedIntArray &arr)

{

rows = arr.rows;

cols = arr.cols;

pa = new int \*[rows];

for (int i = 0; i < rows; i++)

{

pa[i] = new int[cols];

for (int j = 0; j < cols; j++)

{

pa[i][j] = arr.pa[i][j];

}

}

}

// Перевантажена операція додавання

UnsignedIntArray operator+(const UnsignedIntArray &a, const UnsignedIntArray &b)

{

if (a.rows != b.rows || a.cols != b.cols)

{

throw std::invalid\_argument("Matrices must have the same size");

}

UnsignedIntArray result(a.rows, a.cols);

for (int i = 0; i < a.rows; i++)

{

for (int j = 0; j < a.cols; j++)

{

int s1 = a.pa[i][j];

int s2 = b.pa[i][j];

int sum = a.pa[i][j] + b.pa[i][j];

result.pa[i][j] = a.pa[i][j] + b.pa[i][j];

}

}

return result;

}

// Перевантажена операція віднімання

UnsignedIntArray operator-(const UnsignedIntArray &a, const UnsignedIntArray &b)

{

if (a.rows != b.rows || a.cols != b.cols)

{

throw std::invalid\_argument("Matrices must have the same size");

}

UnsignedIntArray result(a.rows, a.cols);

for (int i = 0; i < a.rows; i++)

{

for (int j = 0; j < a.cols; j++)

{

result.pa[i][j] = a.pa[i][j] - b.pa[i][j];

}

}

return result;

}

// Перевантажена операція множення

UnsignedIntArray operator\*(const UnsignedIntArray &a, const UnsignedIntArray &b)

{

if (a.rows != b.cols)

{

throw std::invalid\_argument("The number of columns of the first matrix is equal to the number of rows of the second matrix.");

}

UnsignedIntArray result(a.rows, b.cols);

for (int i = 0; i < a.rows; i++)

{

for (int j = 0; j < b.cols; j++)

{

result.pa[i][j] = 0;

for (int k = 0; k < b.cols; k++)

{

result.pa[i][j] += b.pa[i][k] \* a.pa[k][j];

}

}

}

return result;

}

// Перевантажена операція виведення в потік

ostream &operator<<(ostream &out, const UnsignedIntArray &a)

{

out << "\nArray:" << endl;

for (int i = 0; i < a.rows; i++)

{

for (int j = 0; j < a.cols; j++)

{

out << a.pa[i][j] << "\t";

}

out << "\n";

}

return out;

}

// Перевантажена операція читання з потоку

istream &operator>>(istream &in, UnsignedIntArray &a)

{

cout << "Please enter the elements of the array, row by row:" << endl;

for (int i = 0; i < a.rows; i++)

{

cout << "Row " << i + 1 << ": ";

for (int j = 0; j < a.cols; j++)

{

in >> a.pa[i][j];

cout << a.pa[i][j] << " ";

}

cout << endl;

}

return in;

}

void getLog10(UnsignedIntArray a) // викликаємо конструктор копіювання

{

for (int i = 0; i < a.getRows(); i++)

{

for (int j = 0; j < a.getCols(); j++)

{

if (a[i][j] > 0)

{

a[i][j] = log10(a[i][j]);

}

}

}

}

int main()

{

UnsignedIntArray a(2, 2); // создали матрицу размером 2x2

cin >> a; // считали значения элементов матрицы

cout << "Matrix a:" << a << endl;

// Выполняем операции с матрицами

UnsignedIntArray b(2, 2);

cin >> b; // Считываем матрицу b

cout << "Matrix b:" << b << endl;

try

{

UnsignedIntArray cplus = a + b;

cout << "Matrix a + b:" << cplus << endl;

UnsignedIntArray cminus = a - b;

cout << "Matrix a - b:" << cminus << endl;

UnsignedIntArray cmult = a \* b;

cout << "Matrix a \* b:" << cmult << endl;

}

catch (UnsignedIntArray::OutOfBounds &e)

{

cout << "Error! Index " << e.getIndex() << " outside the array's boundaries." << endl;

}

catch (std::invalid\_argument &e)

{

cout << "Error! " << e.what() << endl;

}

catch (...)

{

cout << "Unknown error!" << endl;

}

// Вызываем функцию для замены положительных элементов на логарифмы

getLog10(a);

// Выводим измененную матрицу

cout << "Matrix a after replacing positive elements by logarithms:" << a << endl;

try

{

// Проверяем границы индексов

cout << "a[2][2] = " << a[2][2] << endl; // Ошибка! Выход за границы массива

}

catch (UnsignedIntArray::OutOfBounds &e)

{

cout << "Error! Index " << e.getIndex() << " outside the array's boundaries." << endl;

}

catch (...)

{

cout << "Unknown error!" << endl;

}

try

{

// Создаем матрицу с неверными параметрами

UnsignedIntArray d(-1, 3); // Ошибка! Неверный размер массива

}

catch (UnsignedIntArray::OutOfBounds &e)

{

cout << "Error when creating a matrix! The number of array elements cannot be: " << e.getIndex() << "." << endl;

}

catch (...)

{

cout << "Unknown error!" << endl;

}

return 0;

}

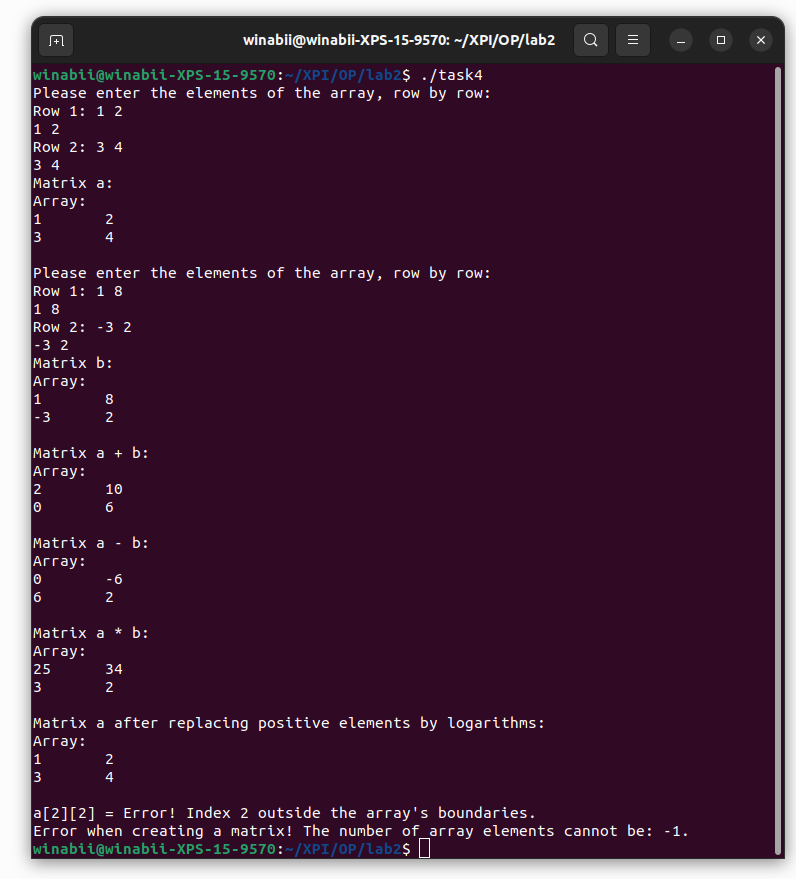


Рис 4 – Приклад роботи четвертої програми

П’яте завдання

Код до п’ятого завдання

#include <iostream>

using std::cout;

using std::endl;

class Counter

{

private:

int x;

static int count;

public:

// Геттер

int getX()

{

return x;

}

static int getCount()

{

return count;

}

Counter(int a)

{

x = a;

count += x;

}

};

// Статичний елемент даних слід визначити й ініціалізувати поза межами класу:

int Counter::count = 0;

int main()

{

Counter c1(23);

Counter c2(3);

Counter c3(10);

Counter c4(-10);

cout << Counter::getCount() << endl;

return 0;

}

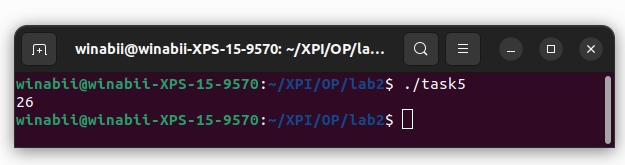


Рис 5 – Приклад роботи п’ятої програми

**Висновок:** Після успішного виконання лабораторної роботи з використанням мови програмування С++, було отримано практичні навички роботи з класами, зокрема у використанні концепцій інкапсуляції, успадкування та поліморфізму. Також було розглянуто імплементацію конструкторів та деструкторів класів, а також патерни проектування, що дозволить бути більш компетентним у розробці складних програмних систем у мові C++.

Вправа для контролю номер 1

Створити клас "Пара рядків" з необхідними конструкторами та функціями доступу.

#include <string>

class PairOfStrings

{

private:

std::string firstString;

std::string secondString;

public:

PairOfStrings() {}

PairOfStrings(const std::string &first, const std::string &second) : firstString(first), secondString(second) {}

void setFirstString(const std::string &str)

{

firstString = str;

}

void setSecondString(const std::string &str)

{

secondString = str;

}

std::string getFirstString() const

{

return firstString;

}

std::string getSecondString() const

{

return secondString;

}

};