МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет

«Харківський політехнічний інститут»

Факультет Комп’ютерних наук та програмної інженерії

Кафедра Інформатики та інтелектуальної власності

ЗВІТ

До лабораторной роботи №2 з дисципліни

«Об’єкто-орієнтоване програмування»

Студент Бородай Д.А

Викладач Івашко А.В.

Харків 2022

**Лабораторна робота №2**

**Тема: «Перевантаження операторів класів»**

**Мета роботи:** навчитись перевантажувати операторів різних видів за допомогою різних способів.

**Завдання:** Визначити клас «Дроб» – **Fraction** у вигляді пари чисельник знаменник. Клас повинен містити кілька конструкторів. Перевантажити оператори додавання, віднімання, множення, поділу, присвоєння та операції відносини. Створити масив об'єктів і передати його в функцію, яка змінює кожен елемент масиву з парним індексом шляхом додавання елемента масиву, що слідує за ним.

## 1 Виконання роботи

Згідно завдання було створено клас Fraction з двома полями даних типу float denominator і numerator, чисельник та знаменник відповідно. Для цього класу було створено два конструктори, один за замовченням із рандомною ініціалізацією полів даних та інший для ініціалізації користувачем. Для роботи з класом були перевантажені оператору потоку вводу та виводу, прості арифметичні та оператори відношень. Для роботи з масивом об’єктів було перевантажено відповідний оператор. А також для зручного виведення такого масиву було створено метод print. Метод task додає до кожного парного масиву елементи що йде за ним. Приклад коду класу наведено нижче.

class Fraction{

float denominator;

float numerator;

public:

Fraction() { (this->denominator = rand() % 50 + 1) / (this->numerator = rand() % 50 + 1); };

Fraction(const float denominator, const float numerator) {

if (numerator != 0)

(this->denominator = denominator) / (this->numerator = numerator);

else

cout << "\tYou whant to brake the math rules? -\_-\n";

};

// stream operators

friend ostream& operator << (ostream& out, const Fraction& ob) { out << ob.denominator << " / " << ob.numerator << " = " << setprecision(10) << ob.denominator / ob.numerator; return out; };

friend istream& operator >> (istream& in, Fraction& ob) { cout << "Denominator --> "; in >> ob.denominator; cout << "Numerator --> "; in >> ob.numerator; return in; };

friend Fraction& operator + (const Fraction& ob1, const Fraction& ob2);

friend Fraction& operator - (const Fraction& ob1, const Fraction& ob2);

friend Fraction& operator / (const Fraction& ob1, const Fraction& ob2);

friend Fraction& operator \* (const Fraction& ob1, const Fraction& ob2);

Fraction& operator = (const Fraction& ob);

//compering operations

bool operator == (const Fraction& ob) { return (denominator / numerator == ob.denominator / ob.numerator);}

bool operator != (const Fraction& ob) { return !(denominator / numerator == ob.denominator / ob.numerator); }

bool operator > (const Fraction& ob) { return (denominator / numerator > ob.denominator / ob.numerator); };

bool operator < (const Fraction& ob) { return (denominator / numerator < ob.denominator / ob.numerator); };

bool operator >= (const Fraction& ob) { return (denominator / numerator >= ob.denominator / ob.numerator); };

bool operator <= (const Fraction& ob) { return (denominator / numerator <= ob.denominator / ob.numerator); };

// tt function

void task(Fraction arr[], int& size);

float operator[](const int i) { return denominator / numerator; };

void print(Fraction arr[], int size);};

Повний код наведено у додатку А.

Код перевантаження простих арифметичних операторів:

Fraction& operator+(const Fraction& ob1, const Fraction& ob2)

{

Fraction result( (ob1.denominator \* ob2.numerator) + (ob2.denominator \* ob1.numerator), ob1.numerator \* ob2.numerator);

return result;

}

Fraction& operator-(const Fraction& ob1, const Fraction& ob2)

{

Fraction result((ob1.denominator \* ob2.numerator) - (ob2.denominator \* ob1.numerator), ob1.numerator \* ob2.numerator);

return result;

}

Fraction& operator/(const Fraction& ob1, const Fraction& ob2)

{

Fraction result(ob2.numerator \* ob1.denominator, ob2.denominator \* ob1.numerator);

return result;

}

Fraction& operator\*(const Fraction& ob1, const Fraction& ob2)

{

Fraction result(ob1.denominator \* ob2.denominator, ob1.numerator \* ob2.numerator);

return result;

}

Fraction& Fraction::operator=(const Fraction& ob)

{

denominator = ob.denominator; numerator = ob.numerator;

return \*this;

}

Повний код наведено у додатку Б.

Приклади роботи перевантажених простих арифметичних операторів :

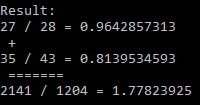


Рисунок 1.1 – Приклад роботи перевантаженого оператора додавання

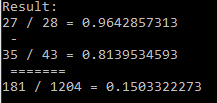


Рисунок 1.2 – Приклад роботи перевантаженого оператора віднімання

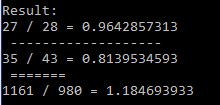


Рисунок 1.3 – Приклад роботи перевантаженого оператора ділення

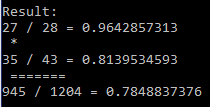


Рисунок 1.4 – Приклад роботи перевантаженого оператора множення

Код перевантажень операторів відношень було реалізовано у класі, що наведено вище. Для перевірки правдивості відношень, користувачу потрібно обрати один елемент з першого масиву об’єктів класу та з другого. При ствердній відповіді на екрані користувача буде написано обрану ним операцію відношення та True, при іншій відповіді False, що означає відповідно «Так, відношення є ствердним» та «Ні, відношення не є ствердним». Приклад роботи одного із відношень, відношення рівності. наведено на рисунку 1.5. У якості першого елементу було обрано дріб 20 / 20 = 1, у якості другого дробу є 40 / 2 = 20.

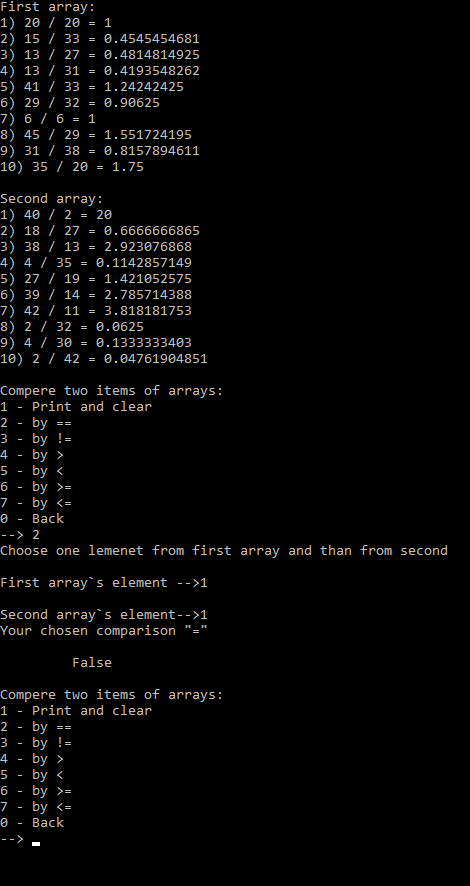


Рисунок 1.5 – Приклад відношення рівності

## 2 Довідка по роботі з програмою для користувача.

Програма собою являє рахівник дробів. Користувач може ввести лише два ірраціональні дроби, та провести ряд арифметичних та відносних операцій. На рисунку 2.1 наведено прості арифметичні операції, а на рисунку 2.2 порівняльні операції.

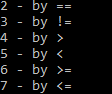


Рисунок 2.1 – Прості арифметичні оператори

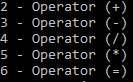


Рисунок 2.2 – Оператори відносності

Для зручного користування надається змога прочитати попередні дії та за необхідності їх видалити.

У восьмому пункті користувач може додати до кожного другого елементу обраного масиву дробів попередній елемент.

**Висновок**

Під час даної лабораторної роботи було закріплено навички з перевантаження операторів різними способами.

**Додаток А**

Header.h

#pragma once

#include <iostream>

#include <ctime>

#include <iomanip>

using namespace std;

class Fraction

{

float denominator;

float numerator;

public:

Fraction() { (this->denominator = rand() % 50 + 1) / (this->numerator = rand() % 50 + 1); };

Fraction(const float denominator, const float numerator) {

if (numerator != 0)

(this->denominator = denominator) / (this->numerator = numerator);

else

cout << "\tYou whant to brake the math rules? -\_-\n";

};

// stream operators

friend ostream& operator << (ostream& out, const Fraction& ob) { out << ob.denominator << " / " << ob.numerator << " = " << setprecision(10) << ob.denominator / ob.numerator; return out; };

friend istream& operator >> (istream& in, Fraction& ob) { cout << "Denominator --> "; in >> ob.denominator; cout << "Numerator --> "; in >> ob.numerator; return in; };

// ar operation

friend Fraction& operator + (const Fraction& ob1, const Fraction& ob2);

friend Fraction& operator - (const Fraction& ob1, const Fraction& ob2);

friend Fraction& operator / (const Fraction& ob1, const Fraction& ob2);

friend Fraction& operator \* (const Fraction& ob1, const Fraction& ob2);

Fraction& operator = (const Fraction& ob);

//compering operations

bool operator == (const Fraction& ob) { return (denominator / numerator == ob.denominator / ob.numerator);}

bool operator != (const Fraction& ob) { return !(denominator / numerator == ob.denominator / ob.numerator); }

bool operator > (const Fraction& ob) { return (denominator / numerator > ob.denominator / ob.numerator); };

bool operator < (const Fraction& ob) { return (denominator / numerator < ob.denominator / ob.numerator); };

bool operator >= (const Fraction& ob) { return (denominator / numerator >= ob.denominator / ob.numerator); };

bool operator <= (const Fraction& ob) { return (denominator / numerator <= ob.denominator / ob.numerator); };

// tt function

void task(Fraction arr[], int& size);

float operator[](const int i) { return denominator / numerator; };

void print(Fraction arr[], int size);

};

class Main

{

public:

void main() const;

};

**Додаток Б**

Source.cpp

#include "Header.h"

void Main::main() const

{

srand(time(NULL));

int size = 10;

Fraction arr[10], newarr[10], result;

int button, fchoose, schoose;

cout << "First array: \n";

arr->print(arr, size);

cout << endl;

cout << "Second array: \n";

newarr->print(newarr, size);

cout << endl;

while (true) {

cout << "1 - Print and clear\n" <<

"2 - Operator (+)\n" <<

"3 - Operator (-)\n" <<

"4 - Operator (/)\n" <<

"5 - Operator (\*)\n" <<

"6 - Operator (=)\n" <<

"7 - Comparison Operators\n" <<

"8 - Add previous to every second element of array\n" <<

"0 - Exit\n" <<

"--> "; cin >> button;

switch (button)

{

case 1:

system("cls");

cout << "First array: \n";

arr->print(arr, size);

cout << endl;

cout << "Second array: \n";

newarr->print(newarr, size);

cout << endl;

break;

case 2:

cout << "\nChoose one lemenet from first array and than from second \n\nFirst array`s element -->"; cin >> fchoose;

cout << "\nSecond array`s element-->"; cin >> schoose;

if (fchoose - 1 > size or fchoose - 1 < 0 or schoose - 1 > size or schoose - 1 < 0) {

cout << "Invalid input\n\n"; break;

}

result = arr[fchoose - 1] + newarr[schoose - 1];

cout << "\nResult: \n" << arr[fchoose-1] << "\n + \n" << newarr[schoose-1] << endl << " ======= "<< endl << result << endl << endl;

break;

case 3:

cout << "Choose one lemenet from first array and than from second \n\nFirst array`s element -->"; cin >> fchoose;

cout << "\nSecond array`s element-->"; cin >> schoose;

if (fchoose - 1 > size or fchoose - 1 < 0 or schoose - 1 > size or schoose - 1 < 0) {

cout << "Invalid input\n\n"; break;

}

result = arr[fchoose - 1] - newarr[schoose - 1];

cout << "\nResult: \n" << arr[fchoose - 1] << "\n - \n" << newarr[schoose - 1] << endl << " ======= " << endl << result << endl << endl;

break;

case 4:

cout << "Choose one lemenet from first array and than from second \n\nFirst array`s element -->"; cin >> fchoose;

cout << "\nSecond array`s element-->"; cin >> schoose;

if (fchoose - 1 > size or fchoose - 1 < 0 or schoose - 1 > size or schoose - 1 < 0) {

cout << "Invalid input\n\n"; break;

}

result = arr[fchoose - 1] / newarr[schoose - 1];

cout << "\nResult: \n" << arr[fchoose - 1] << "\n ------------------- \n" << newarr[schoose - 1] << endl << " ======= " << endl << result << endl << endl;

break;

case 5:

cout << "Choose one lemenet from first array and than from second \n\nFirst array`s element -->"; cin >> fchoose;

cout << "\nSecond array`s element-->"; cin >> schoose;

if (fchoose - 1 > size or fchoose - 1 < 0 or schoose - 1 > size or schoose - 1 < 0) {

cout << "Invalid input\n\n"; break;

}

result = arr[fchoose - 1] \* newarr[schoose - 1];

cout << "\nResult: \n" << arr[fchoose - 1] << "\n \* \n" << newarr[schoose - 1] << endl << " ======= " << endl << result << endl << endl;

break;

case 6:

cout << "Choose one lemenet from first array and than from second \n\nFirst array`s element -->"; cin >> fchoose;

cout << "\nSecond array`s element-->"; cin >> schoose;

if (fchoose - 1 > size or fchoose - 1 < 0 or schoose - 1 > size or schoose - 1 < 0) {

cout << "Invalid input\n\n"; break;

}

arr[fchoose - 1] = newarr[schoose - 1];

break;

case 7:

system("cls");

cout << "First array: \n";

arr->print(arr, size);

cout << endl;

cout << "Second array: \n";

newarr->print(newarr, size);

cout << endl;

while (button != 0) {

cout << "Compere two items of arrays: \n"

<< "1 - Print and clear\n"

<< "2 - by == \n"

<< "3 - by != \n"

<< "4 - by > \n"

<< "5 - by < \n"

<< "6 - by >= \n"

<< "7 - by <= \n"

<< "0 - Back \n"

<< "--> "; cin >> button;

switch (button)

{

case 1:

system("cls");

cout << "First array: \n";

arr->print(arr, size);

cout << endl;

cout << "Second array: \n";

newarr->print(newarr, size);

cout << endl;

break;

case 2:

cout << "Choose one lemenet from first array and than from second \n\nFirst array`s element -->"; cin >> fchoose;

cout << "\nSecond array`s element-->"; cin >> schoose;

if (fchoose - 1 > size or fchoose - 1 < 0 or schoose - 1 > size or schoose - 1 < 0) {cout << "\nInvalid input!\n\n"; break;}

cout << "Your chosen comparison \"=\" \n";

if (arr[fchoose - 1] == newarr[schoose - 1])

cout << "\n\t True \t\n\n";

else

cout << "\n\t False \t\n\n";

break;

case 3:

cout << "Choose one lemenet from first array and than from second \n\nFirst array`s element -->"; cin >> fchoose;

cout << "\nSecond array`s element-->"; cin >> schoose;

if (fchoose - 1 > size or fchoose - 1 < 0 or schoose - 1 > size or schoose - 1 < 0) { cout << "\nInvalid input!\n\n"; break; }

cout << "\nYour chosen comparison \"!=\" \n";

if (arr[fchoose - 1] != newarr[schoose - 1])

cout << "\n\t True \t\n\n";

else

cout << "\n\t False \t\n\n";

break;

case 4:

cout << "Choose one lemenet from first array and than from second \n\nFirst array`s element -->"; cin >> fchoose;

cout << "\nSecond array`s element-->"; cin >> schoose;

if (fchoose - 1 > size or fchoose - 1 < 0 or schoose - 1 > size or schoose - 1 < 0) { cout << "\nInvalid input!\n\n"; break; }

cout << "\nYour chosen comparison \">\" \n";

if (arr[fchoose - 1] > newarr[schoose - 1])

cout << "\n\t True \t\n\n";

else

cout << "\n\t False \t\n\n";

break;

case 5:

cout << "Choose one lemenet from first array and than from second \n\nFirst array`s element -->"; cin >> fchoose;

cout << "\nSecond array`s element-->"; cin >> schoose;

if (fchoose - 1 > size or fchoose - 1 < 0 or schoose - 1 > size or schoose - 1 < 0) { cout << "\nInvalid input!\n\n"; break; }

cout << "\nYour chosen comparison \"<\" \n";

if (arr[fchoose - 1] < newarr[schoose - 1])

cout << "\n\t True \t\n\n";

else

cout << "\n\t False \t\n\n";

break;

case 6:

cout << "Choose one lemenet from first array and than from second \n\nFirst array`s element -->"; cin >> fchoose;

cout << "\nSecond array`s element-->"; cin >> schoose;

if (fchoose - 1 > size or fchoose - 1 < 0 or schoose - 1 > size or schoose - 1 < 0) { cout << "\nInvalid input!\n\n"; break; }

cout << "\nYour chosen comparison \">=\" \n";

if (arr[fchoose - 1] >= newarr[schoose - 1])

cout << "\n\t True \t\n\n";

else

cout << "\n\t False \t\n\n";

break;

case 7:

cout << "Choose one lemenet from first array and than from second \n\nFirst array`s element -->"; cin >> fchoose;

cout << "\nSecond array`s element-->"; cin >> schoose;

if (fchoose - 1 > size or fchoose - 1 < 0 or schoose - 1 > size or schoose - 1 < 0) { cout << "\nInvalid input!\n\n"; break; }

cout << "\nYour chosen comparison \"<=\" \n";

if (arr[fchoose - 1] <= newarr[schoose - 1])

cout << "\n\t True \t\n\n";

else

cout << "\n\t False \t\n\n";

break;

case 0:

system("cls");

break;

default:

break;

}

}

break;

case 8:

cout << "\nSelect the array to which this operation will be applied \n1 - First array \n2 - Second array \n--> "; cin >> button;

if (button != 1 and button != 2) { cout << "\nInvalid input!\n\n"; break;}

if (button == 1)

arr->task(arr, size);

else if (button == 2)

newarr->task(newarr, size);

cout << "First array: \n";

arr->print(arr, size);

cout << endl;

cout << "Second array: \n";

newarr->print(newarr, size);

cout << endl;

break;

case 0:

system("cls");

exit(0);

break;

default:

break;

}

}

}

void Fraction::print(Fraction arr[], int size)

{

for (size\_t i = 0; i < size; i++)

cout << i+1 << ") " << arr[i] << endl;

}

Fraction& operator+(const Fraction& ob1, const Fraction& ob2)

{

Fraction result( (ob1.denominator \* ob2.numerator) + (ob2.denominator \* ob1.numerator), ob1.numerator \* ob2.numerator);

return result;

}

Fraction& operator-(const Fraction& ob1, const Fraction& ob2)

{

Fraction result((ob1.denominator \* ob2.numerator) - (ob2.denominator \* ob1.numerator), ob1.numerator \* ob2.numerator);

return result;

}

Fraction& operator/(const Fraction& ob1, const Fraction& ob2)

{

Fraction result(ob2.numerator \* ob1.denominator, ob2.denominator \* ob1.numerator);

return result;

}

Fraction& operator\*(const Fraction& ob1, const Fraction& ob2)

{

Fraction result(ob1.denominator \* ob2.denominator, ob1.numerator \* ob2.numerator);

return result;

}

Fraction& Fraction::operator=(const Fraction& ob)

{

denominator = ob.denominator; numerator = ob.numerator;

return \*this;

}

void Fraction::task(Fraction arr[], int& size)

{

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

if ((i+1) % 2 == 0 and i - 1 >= 0)

arr[i] = arr[i] + arr[i - 1];

}