Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра ЕОМ



**Звіт**

з лабораторної роботи №5

з дисципліни: “Програмування, ч.2 (ООП)”

на тему: “ перевантаження операторів”

Варіант 19

Виконав: ст.гр. КІ-15

Мармура В.І.

Прийняла: Викладач

Козак Н.Б.

Львів

2020

Мета: : познайомитися із перевантаженням операторів.

Теоретичні відомості :

Щоб використовувати операцію над об'єктами класів, ця операція повинна бути перевантажена, але є два виключення. Операції присвоювання (=) і взяття адреси (&) створюються в класі автоматично за замовчуванням, тому їх можна використовувати без явного перевантаження. За замовчуванням операція присвоювання зводиться до побітового копіювання даних-елементів класу. Проте таке побітове копіювання небезпечне для класів з елементами, що вказують на динамічно виділені області пам'яті, масиви, рядки, оскільки в цьому випадку відбувається копіювання не даних (глибоке копіювання), а лише вказівників на дані (поверхневе копіювання). Для таких класів слід явно перевантажувати операцію присвоювання і здійснювати у ній глибоке копіювання. Операція адресації також може бути використана з об'єктами будь-яких класів без перевантаження. Вона просто повертає адресу об'єкта в пам'яті. Але операцію адресації можна також і перевантажувати. Перевантаження операцій підпорядковується наступним правилам:

• При перевантаженні зберігаються кількість аргументів, пріоритети операцій та правила асоціації, що використовуються у стандартних типах даних;

• Для стандартних типів даних операції не підлягають перевизначенню;

• Перевантажена функція-оператор не може мати параметрів по замовчуванню, не успадковується та не може бути визначеною як static;

• Функція-оператор може бути визначена трьома способами – метод класу, дружня функція або звичайна функція. В останніх двох випадках вона повинна приймати хоча б один аргумент, що має тип класу, вказівника або посилання на клас. При перевантаженні операцій ( ), [], -> та = функція перевантаження операції може бути оголошена лише як метод класу. Для інших операцій функції перевантаження операцій можуть не бути методами класу. \

Завдання (Варіант 19) -

Розширити функціональність розроблених у 4 лабораторній роботі класів за допомогою операторів, що задані варіантом та оператора присвоювання. Конкретні функції операторів реалізувати на власний розсуд (крім оператора присвоювання). Організувати виведення та введення даних за допомогою класів-потоків сin, cout та перевантажених операторів вводу/виводу. Написати програму, яка демонструє роботу з об'єктами цього класу.

Код для вирішення задачі даного варіанту -

#include <iostream>

#include <String>

using namespace std;

void clear() {

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

class Ship {

public:

void setShipName(string Name) {

sName = Name;

}

void setShipPass(int passengers) {

sPass = passengers;

}

void setShipFuel(int fuel) {

sFuel = fuel;

}

void setShipValue(int value) {

sValue = value;

}

void setShipMilesValue(int cost) {

sMileValue = cost;

}

void setShipFuelPerMile(int fuel) {

sFuelPerMile = fuel;

}

string getShipName() {

return sName;

}

int getShipPass() {

return sPass;

}

int getShipFuel() {

return sFuel;

}

int getShipValue() {

return sValue;

}

int getShipMilesValue() {

return sMileValue;

}

int getShipFuelPerMile() {

return sFuelPerMile;

}

bool Move(int miles) {

sValue -= sMileValue \* miles;

sMiles += miles;

sFuel -= sFuelPerMile \* miles;

if (sFuel > 0) {

return true;

}

else if (sFuel <= 0) {

cout << "Fuel tank empty, ship had stucked in ocean..." << endl;

return false;

}

}

void PortOut(int passengers) {

sPass -= passengers;

}

void PortIn(int passengers) {

sPass += passengers;

}

void State() {

clear();

cout << "Ship Name - " << this->sName << endl;

cout << "Ship Passengers - " << this->sPass << endl;

cout << "Ship Fuel - " << this->sFuel << endl;

cout << "Ship Value - " << this->sValue << endl;

cout << "Ship Miles - " << this->sMiles << endl;

clear();

}

Ship(string name, int passengers, int fuel, int value, int valuepermile, int fuelpermile) {

sName = name;

sPass = passengers;

sFuel = fuel;

sValue = value;

sMileValue = valuepermile;

sMiles = 0;

sFuelPerMile = fuelpermile;

}

friend bool operator+= (Ship ship1, Ship ship2) {

ship1.sFuel += ship2.getShipFuel();

cout << "Ship 1 fuel = " << ship1.sFuel << endl;

cout << "Ship 2 fuel = " << ship2.getShipFuel() << endl;

return true;

}

friend bool operator-= (Ship ship1, Ship ship2) {

ship1.sFuel -= ship2.getShipFuel();

cout << "Ship 1 fuel = " << ship1.sFuel << endl;

cout << "Ship 2 fuel = " << ship2.getShipFuel() << endl;

return false;

}

Ship& operator= (const Ship& ship2) {

if (this == &ship2) {

return \*this;

}

sFuel = ship2.sFuel;

sPass = ship2.sPass;

return \*this;

}

private:

string sName;

int sPass, sFuel, sValue, sMileValue, sMiles, sFuelPerMile;

};

bool operator == (Ship ship1, Ship ship2) {

if (ship1.getShipFuel() == ship2.getShipFuel() && ship1.getShipName() == ship2.getShipName() && ship1.getShipMilesValue() == ship2.getShipMilesValue()) {

return true;

}

else return false;

}

bool operator != (Ship ship1, Ship ship2) {

if (ship1.getShipFuel() == ship2.getShipFuel() && ship1.getShipName() == ship2.getShipName() && ship1.getShipMilesValue() == ship2.getShipMilesValue()) {

return false;

}

else return true;

}

bool operator > (Ship ship1, Ship ship2) {

if (ship1.getShipFuel() > ship2.getShipFuel() && ship1.getShipPass() > ship2.getShipPass()) {

return true;

}

else return false;

}

bool operator < (Ship ship1, Ship ship2) {

if (ship1.getShipFuel() < ship2.getShipFuel() && ship1.getShipName() < ship2.getShipName() && ship1.getShipMilesValue() < ship2.getShipMilesValue()) {

return true;

}

else return false;

}

int main() {

string sName, sName1;

int sPassengers, sFuel, sValue, sMileValue, sFuelPerMile;

int sPassengers1, sFuel1, sValue1, sMileValue1, sFuelPerMile1;

cout << "Enter a first ship's name - ";

cin >> sName;

cout << endl << "Enter a ship's passengers on board - ";

cin >> sPassengers;

cout << endl << "Enter a ship's fuel on board - ";

cin >> sFuel;

cout << endl << "Enter a ship's value - ";

cin >> sValue;

cout << endl << "Enter a ship's usage value per 1 mile - ";

cin >> sMileValue;

cout << endl << "Enter a ship's fuel usage per 1 mile - ";

cin >> sFuelPerMile;

Ship myShip1(sName, sPassengers, sFuel, sValue, sMileValue, sFuelPerMile);

cout << endl << endl;

cout << endl << "First ship state - " << endl;

myShip1.State();

cout << "Enter a second ship's name - ";

cin >> sName1;

cout << endl << "Enter a ship's passengers on board - ";

cin >> sPassengers1;

cout << endl << "Enter a ship's fuel on board - ";

cin >> sFuel1;

cout << endl << "Enter a ship's value - ";

cin >> sValue1;

cout << endl << "Enter a ship's usage value per 1 mile - ";

cin >> sMileValue1;

cout << endl << "Enter a ship's fuel usage per 1 mile - ";

cin >> sFuelPerMile1;

Ship myShip2(sName1, sPassengers1, sFuel1, sValue1, sMileValue1, sFuelPerMile1);

cout << endl << "Second ship state - " << endl;

myShip2.State();

if (myShip1==myShip2) {

cout << "Ships are same" << endl;

}

if (myShip1>myShip2) {

cout << "First ship is more loaded than second" << endl;

}

myShip1 += myShip2;

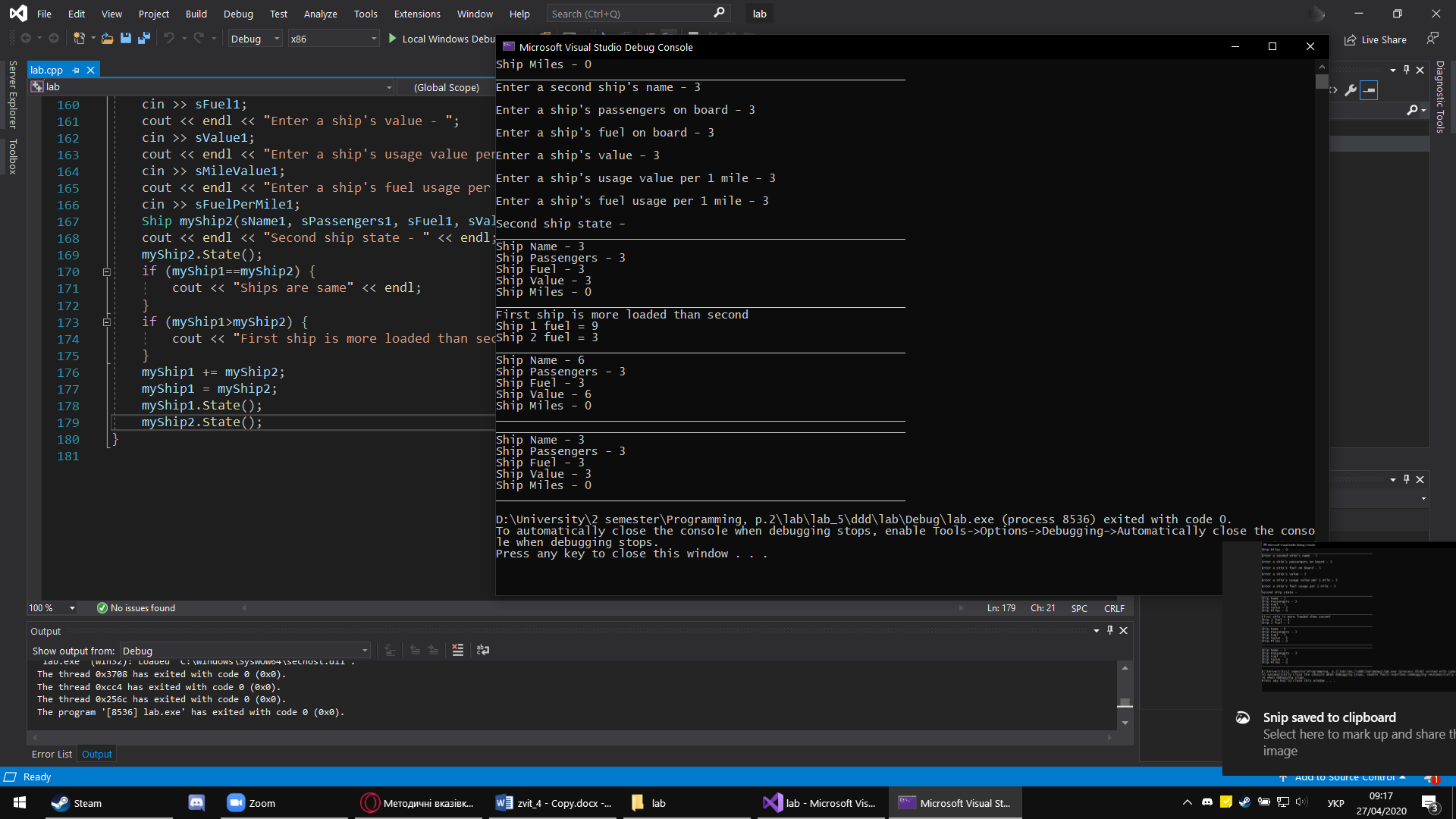
myShip1 = myShip2;

myShip1.State();

myShip2.State();

}

Скріншот виконання програми –



Висновок : в даній лабораторній роботі я опрацював роботу з классами, об’єктами классів, застосуванням їх і використанням геттерів і сеттерів.