

Lab.5.1(1)

#include <iostream>

#include <string>

using namespace std;

class Person

{

public:

string name;

int age;

string lastName;

};

string FirstLastName(Person\* person)

{

return person->name + " " + person->lastName;

}

void print(Person\* person)

{

cout << FirstLastName(person) << " is " << person->age << " years old" << endl;

}

int main()

{

Person person;

person.name = "Marry";

person.age = 20;

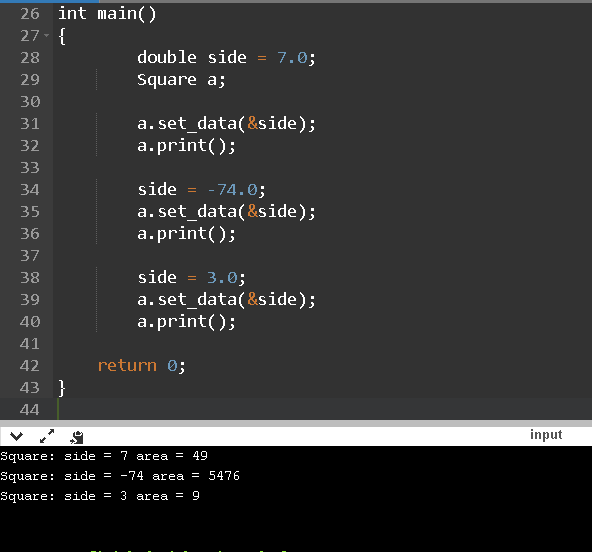
person.lastName = "Harson";

cout << "Meet " << person.name << "\n";

print(&person); printf("\n");

return 0;

}



Lab. 5.1(2)

#include <iostream>

#include<cstdlib>

#include<string>

#include <algorithm>

using namespace std;

class Square

{

private:

double side;

double area;

public:

void set\_data(double\* side)

{

this->side = \*side;

area = this->side \* this->side;

}

void print()

{

cout << "Square: side = " << side << " area = " << area << endl;

}

};

int main()

{

double side = 7.0;

Square a;

a.set\_data(&side);

a.print();

side = -74.0;

a.set\_data(&side);

a.print();

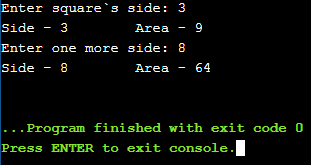
side = 3.0;

a.set\_data(&side);

a.print();

return 0;

}

****

Lab.5.1(3)

#include <iostream>

#include<string>

#include<cmath>

using namespace std;

class AdHocSquare

{

private:

double side;

public:

AdHocSquare(double side)

{

this->side = side;

}

double get\_area()

{

return (this->side \* this->side);

}

void print()

{

cout << "Side - " << this->side << "\t" << "Area - " << get\_area() << endl;

}

};

class LazySquare

{

public:

LazySquare(double side, double prev\_side)

{

this->side = side;

this->area = pow(this->side, 2);

if (prev\_side == this->side)

this->side\_changed = false;

else

this->side\_changed = true;

}

void print()

{

if (!side\_changed)

cout << "We don`t need to do it, you have it higher" << endl;

else

cout << "Side - " << this->side << "\t" << "Area - " << this->area << endl;

}

private:

double side;

double area;

bool side\_changed;

};

int main()

{

double side, side1;

cout << "Enter square`s side: ";

cin >> side;

AdHocSquare square(side);

square.print();

cout << "Enter one more side: ";

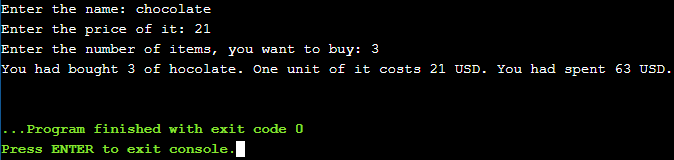
cin >> side1;

LazySquare square1(side1, side );

square1.print();

return 0;

}

****

Lab.5.1(4)

#include <iostream>

#include<string>

#include<cmath>

using namespace std;

class ShopItemOrder

{

private:

string name;

float price, total\_price;

int amount;

public:

ShopItemOrder(string name, float price, int amount)

{

this->name = name;

this->price = price;

this->amount = amount;

this->total\_price = price \* amount;

}

void print()

{

cout << "You had bought " << this->amount << " of " << this->name << ". One unit of it costs " << this->price << " USD. You had spent " << this->total\_price << " USD." << endl;

}

};

int main()

{

string name;

float price;

int amount;

cout << "Enter the name: ";

cin.ignore();

getline(cin, name);

cout << "Enter the price of it: ";

cin >> price;

cout << "Enter the number of items, you want to buy: ";

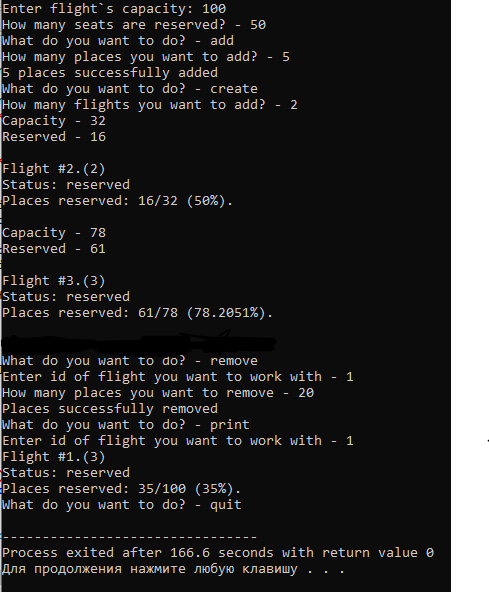
cin >> amount;

ShopItemOrder order(name, price, amount);

order.print();

return 0;

}



Lab. 5.3(1)-5.3(3)

#include <iostream>

#include <string>

using namespace std;

class FlightBooking

{

public:

//Конструктори

FlightBooking() // Порожній конструктор для можливості об яви заголовку об єкту класу

{

}

FlightBooking(int capacity, int reserved) // Конструктор

{

idCount++; // Статичний лічильник об єктів

this->id = this->idCount; // За допомогою статичного лічильника маємо примітивний генератор індексів

this->capacity = capacity;

this->reserved = reserved;

}

void set\_reserved(int reserved) // Метод для зміни кількості зарезервованих місць

{

this->reserved = reserved;

}

// Гет

int get\_capacity()

{

return this->capacity;

}

int get\_reserved()

{

return this->reserved;

}

int get\_id()

{

return this->id;

}

static int get\_count()

{

return idCount;

}

double get\_percantage()

{

return (double(this->reserved) / double(this->capacity)) \* 100;

}

// ---------------------------------

void cancel() // Відміна рейсу

{

this->reserved = NULL; // ßêùî Якщо кількість зарезервованих місць=0, то рейс відміняється

}

void print() // Виведення рейсу

{

if (this->reserved != NULL) // Якщо рейс не відміненний

cout << "Flight #" << this->id << ".(" << get\_count() << ")\nStatus: reserved\nPlaces reserved: " << this->reserved << "/" << this->capacity << " (" << get\_percantage() << "%)." << endl;

else

cout << "Flight #" << this->id << " is canceled." << endl;

}

private:

static int idCount; // Статичний лічильник генератора індексів

int id;

int capacity;

int reserved;

};

int FlightBooking::idCount = 0; // Початкове значення для статичної змінної

int main()

{

bool created = false; // Змінна, яка визначає чи були створені нові рейси

FlightBooking\* booking1; // Заголовок для майбутнього масиву об єктів класу(створенний для коректності роботи областей видимості)

int capacity, reserved, newFlight, ind;

string command;

// Введення даних по стартовому рейсу (обов язкове)

cout << "Enter flight`s capacity: ";

cin >> capacity;

cout << "How many seats are reserved? - ";

cin >> reserved;

FlightBooking booking(capacity, reserved);

while (command != "quit") // Цикл, що продовжується поки не буде команди. вийти.

{

cout << "What do you want to do? - "; // Вибір команди

cin >> command;

// Список доступних команд

if (command == "add" || command == "print" || command == "cancel" || command == "delete" || command == "create" || command == "remove")

{

if (command == "create") // Створює нові рейси

{

cout << "How many flights you want to add? - ";

cin >> newFlight; // Кількість нових рейсів

if (newFlight <= 10) // Обмеження по кількості нових рейсів

{

int cap, res;

booking1 = new FlightBooking[newFlight]; // створення динамічного масиву об єктів розробленого класу

for (int i = 0; i < newFlight; i++) // Заповненння даними по кожному з рейсів

{

cout << "Capacity - ";

cin >> cap;

cout << "Reserved - ";

cin >> res;

booking1[i] = FlightBooking(cap, res);

cout << endl;

booking1[i].print();

cout << endl;

}

created = true; // Змінна статусу мулевої змінної

}

else

cout << "That`s too much" << endl; // якщо занадто багато рейсів

}

if (command == "print") // команда виводу

{

if (!created) //Якщо не було створенно нових рейсів, то вивести інформацію лише по старому booking.print();

else // якщо були створенні

{

cout << "Enter id of flight you want to work with - "; // вибір конкретного рейсу

cin >> ind;

if (ind == booking.get\_id()) //якщо вибранний перший(стартовий рейс) booking.print();

else

for (int i = 0; i < newFlight; i++) // цикл по массиву рейсів

if (ind == booking1[i].get\_id())

{

booking1[i].print();

break;

}

}

}

if (command == "add") // резервування додаткових місць в рейсі

{

int addition;

if (!created) // якщо не було створенно нових рейсів то робота ведеться лише з стартовим

{

cout << "How many places you want to add? - "; // кількість нових місць

cin >> addition;

if (booking.get\_reserved() + addition <= capacity && booking.get\_reserved() != NULL) // Обмеження, що не дозволяє вести більше місць, чим в літаку

{

booking.set\_reserved(booking.get\_reserved() + addition); // змнна значення зарезервованих місць

cout << addition << " places successfully added" << endl;

}

else

cout << "Flight #" << booking.get\_id() << " is overflowing" << endl; // якщо рейс стане переповненим

}

else

{

cout << "Enter id of flight you want to work with - "; // вибрати рейс, з я ким буде вестись робота

cin >> ind;

// Повтор попередньої процедури

if (ind == booking.get\_id())

{

cout << "How many places you want to add? - ";

cin >> addition;

if (booking.get\_reserved() + addition <= capacity && booking.get\_reserved() != NULL)

{

booking.set\_reserved(booking.get\_reserved() + addition);

cout << addition << " places successfully added" << endl;

}

else

cout << "Flight #" << booking.get\_id() << " is overflowing" << endl;

}

else

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

if (booking1[i].get\_id() == ind)

{

cout << "How many places you want to add? - ";

cin >> addition;

if (booking1[i].get\_reserved() + addition <= booking1[i].get\_capacity() && booking1[i].get\_reserved() != NULL)

{

booking1[i].set\_reserved(booking1[i].get\_reserved() + addition);

cout << addition << " places successfully added" << endl;

}

else

cout << "Flight #" << booking1[i].get\_id() << " is overflowing" << endl;

}

}

}

if (command == "cancel") // відміняє рейс

{

if (!created) // якщо не було створенно нових рейсів то робота ведеться лише з стартовим

{

booking.cancel();

cout << "Flight #" << booking.get\_id() << " successfully canceled" << endl;

}

else

{

cout << "Enter id of flight you want to work with - "; // // вибрати рейс, з я ким буде вестись робота

cin >> ind;

if (ind == booking.get\_id())

{

booking.cancel();

cout << "Flight #" << booking.get\_id() << " successfully canceled" << endl;

}

else

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

if (booking1[i].get\_id() == ind)

{

booking1[i].cancel();

cout << "Flight #" << booking1[i].get\_id() << " successfully canceled" << endl;

}

}

}

if (command == "remove") // видалення місць

{

int deleting;

if (!created) // якщо не було створенно нових рейсів то робота ведеться лише з стартовим

{

cout << "How many places you want to remove - ";

cin >> deleting;

if (booking.get\_reserved() - deleting > 0 && booking.get\_reserved() != NULL) // якщо місця не будуть від ємними та рейси не відмінили

{

booking.set\_reserved(booking.get\_reserved() - deleting);

cout << "Places successfully removed" << endl;

}

else

cout << "Error" << endl; якщо місця були від ємними чи рейси не відмінили }

else

{

cout << "Enter id of flight you want to work with - "; // Вибір рейсу, з яким будемо працювти

cin >> ind;

if (ind == booking.get\_id())

{

cout << "How many places you want to remove - ";

cin >> deleting;

if (booking.get\_reserved() - deleting > 0 && booking.get\_reserved() != NULL)

{

booking.set\_reserved(booking.get\_reserved() - deleting);

cout << "Places successfully removed" << endl;

}

else

cout << "Error" << endl;

}

else

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

if (booking1[i].get\_id() == ind)

{

cout << "How many places you want to remove - ";

cin >> deleting;

if (booking1[i].get\_reserved() - deleting > 0 && booking1[i].get\_reserved() != NULL)

{

booking1[i].set\_reserved(booking1[i].get\_reserved() - deleting);

cout << "Places successfully removed" << endl;

}

else

cout << "Error" << endl;

}

}

}

}

else

if (command != "quit") // Якщо команди немає в списку доступних і команда не є командою виходу, виводиться помилка

cout << "You cannot do this" << endl;

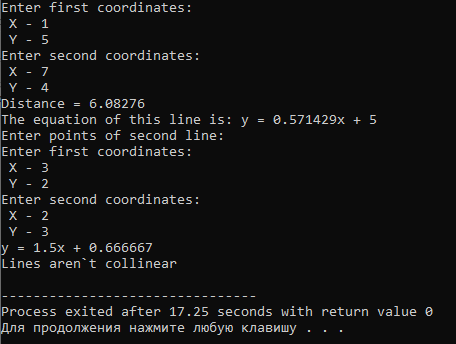
}

if (created) // якщо були створені нові рейси, то звільнити пам ять

delete[] booking1;

return 0;

}



Lab.5.3(8)-5.3(10)

#include <iostream>

#include <string>

#include<cmath>

using namespace std;

class Point2D // точка

{

public:

// конструктор

Point2D(double x, double y)

{

this->x = x;

this->y = y;

}

// методи

double get\_x() // получить Х

{

return this->x;

}

double get\_y() // поулучить У

{

return this->y;

}

double distance(Point2D sec) // Відстань між точками

{

return sqrt(pow((sec.get\_x() - this->x), 2) + pow((sec.get\_y() - this->y), 2));

}

private:

double x;

double y;

};

class Line2D // лінія

{

public:

Line2D(Point2D p1, Point2D p2) // конструктор

{

this->slope = p2.get\_y() / p2.get\_x(); // k

this->y\_intercept = p1.get\_y() / p1.get\_x(); // b

this->coor\_x = p2.get\_x() - p1.get\_x(); // координата х лінії

this->coor\_y = p2.get\_y() - p1.get\_y(); // координата у лінії

}

double get\_coor\_y() // отримати у

{

return this->coor\_y;

}

double get\_coor\_x() // отримати х

{

return this->coor\_x;

}

bool isCollinear(Line2D l1) // перевірка 2 ліній на колінеарність

{

if ((l1.get\_coor\_x() / this->coor\_x) == (l1.get\_coor\_y() / this->coor\_y))

return true;

else

return false;

}

void print() // рівняння прямої (y = kx + b)

{

if (this->y\_intercept > 0) // якщо b > 0

cout << "y = " << this->slope << "x + " << this->y\_intercept << endl;

if (this->y\_intercept == 0) // якщо b == 0

cout << "y = " << this->slope << "x" << endl;

if (this->y\_intercept < 0) // якщо b < 0

cout << "y = " << this->slope << "x " << this->y\_intercept << endl;

}

void printCol(Line2D l1) // вивести чи колінеарні лінії

{

if (isCollinear(l1))

cout << "Lines are collinear" << endl;

else

cout << "Lines aren`t collinear" << endl;

}

private:

double slope;

double y\_intercept;

double coor\_x;

double coor\_y;

};

int main()

{

double x1, x2, y1, y2;

// координати першої точки

cout << "Enter first coordinates:\n X - ";

cin >> x1;

cout << " Y - ";

cin >> y1;

// координати другої точки

cout << "Enter second coordinates:\n X - ";

cin >> x2;

cout << " Y - ";

cin >> y2;

Point2D pnt1(x1, y1), pnt2(x2, y2);

Line2D line1(pnt1, pnt2); // лінія з цих точок

cout << "Distance = " << pnt1.distance(pnt2) << endl; // вивод довжини лінії(відстань між точками)

cout << "The equation of this line is: ";

line1.print(); // вивод рівняння прямої

// ввод точок для другої лінії

cout << "Enter points of second line:" << endl;

cout << "Enter first coordinates:\n X - ";

cin >> x1;

cout << " Y - ";

cin >> y1;

cout << "Enter second coordinates:\n X - ";

cin >> x2;

cout << " Y - ";

cin >> y2;

Point2D pnt3(x1, y1), pnt4(x2, y2);

Line2D line2(pnt3, pnt4);

line2.print(); // вивод рівняння другої прямої

line1.printCol(line2); // чи колінеарні

return 0;

}