#pragma once

#include<iostream>

#include<Windows.h>

#include <time.h>

#include<fstream>

#include <cstring>

#include <string>

#include <algorithm>

#include "Header.h"

class Student

{

char\* name;

public:

explicit Student(const char\* name) : name{ new char[strlen(name) + 1] } //запрет неявный вызов конструктора с 1 аргументом - explicit

//2 аргумента, один из который по умолчанию - тоже самое

{

strcpy\_s(this->name, strlen(name) + 1, name);

}

Student(Student&& other) //конструктор перемещения (забирает все себе, а объект обнуляет). && - это ссылка на правостороннее значение

{

name = other.name;

other.name = nullptr;

}

~Student()

{

delete[] name;

}

Student(const Student& other) :Student(other.name) //здесь могу вызвать только конст методы

{

}

Student& operator=(const Student& other) // нужна эта перегрузка, только если есть указатели

{

if (this != &other) //проверка на присваивание самого себе

{

delete[]name;

name = new char[strlen(other.name) + 1];

strcpy\_s(name, strlen(other.name) + 1, other.name);

}

return \*this; //возврат ссылки на объект для многократного присваивания

}

bool operator< (const Student& right) const

{

return strcmp(name, right.name) < 0;

}

bool operator== (const Student& other) const // параметр 1, тк первым идет this const в методе говорит компилятору, что мы ничего не меняем

{

return strcmp(name, other.name) == 0;

}

friend bool operator!= (const Student& left, const Student& right); //дружественная функция, но не метод класса

const char\* getName()

{

name = 0; //могу поменять имя

return name;

}

const char\* getName() const //сообщаем компилятору о константности метода const после, а const до - это тип ворвращ значения

{

name[0] = 'a'; //не могу поменять имя, могу поменять отдельные символа

return name;

}

};

//не является методом студента - это глобальная функция. дб 2 параметра

// нет прямого доступа к полям

bool operator!= (const Student& left, const Student& right) //мб и методом внутри класса

{

return strcmp(left.getName(), right.getName()) != 0;

}

std::ostream& operator<<(std::ostream& os, const Student& student) //только функция глоб

{

os << student.getName();

return os;

}

std::istream& operator<<(std::istream& is, Student& student)

{

char buffer[200];

is >> buffer;

//что-тоделаем

return is;

}

class Group

{

Student\*\* students;

int capaciti = 10;

int count = 0;

public:

Group() = default; //создать реализацию по умолчанию

Group() = delete; // удалить конструктор или деструктор, чтобы никто не смог присвоить новое, только читать

Group()

{

students = new Student \* [capaciti];

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

students[i] = nullptr;

}

~Group()

{

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

delete students[i]; // могут возникунуть повторные удаления

delete[] students;

}

friend Student; //объявила студент дружественным, но группа к закрытым полям студента не имеет доступа.

//для 2стор надо в обоих классах объявлять

void creatStudent(const char\* name)

{

Student\* stud = new Student(name);

addStudents(stud);

}

// private:

void addStudents(Student\* stud)

{

students[count] = stud;

count++;

}

};

void f(Student s)

{

}

int main()

{

using namespace std;

/\*

Student\* stud = new Student();

Group group;

group.addStudents(stud);

// здесь указатель не удаляем, т.к. он удален в группе

Student stud2 ("Ira");

Group group;

group.addStudents(&stud2);

//здесь deete не нужен. он нужен только когда создан при помощи new

group.creatStudent("valya");

Student vova = "Vova"; //неявный вызов конструктора с 1 аргументом либо в функцию передать. это можно запретить - explicit

Student stud("Petya");

cout << stud->getName();

const Student stud("Vasya");

cout << stud->getName();

Student s1("1"), s2("2");

s1 == s2;

const Student stud("Petya");

cout << stud << stud2<< 1;

ofstream file("out.txt"); // ofstream наследник ostream запись в файл

file << stud << stud2;

\*/

Student students[] = { Student("Alex"),Student("Yana"),Student("Danil"), Student("Boris") };

sort(students, students + 4);

for (int i = 0; i < 4; i++)

cout << students[i] << ' ';

Student a("a");

Student b("b");

Student c("c");

a = b = c;

f(a);

f(Student("Temp")); //временный объект будет сразу уничтожен после работы функции

Student d = move(a); //эта функция библ возвращает правостороннюю ссылку, присваивание без копирования объекта

Group g1;

Group g2;

g2 = g1; //еслю удалю метод, то не смогу так написать

}