#pragmaonce

#include<iostream>

#include<Windows.h>

#include<time.h>

#include<fstream>

#include<cstring>

#include<string>

#include<algorithm>

#include"Header.h"

classStudent

{

char\* name;

public:

explicit Student(constchar\* name) : name{ newchar[strlen(name) + 1] } //запретнеявныйвызовконструкторас 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) //здесьмогувызватьтолькоконстметоды

{

}

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

}

booloperator< (constStudent&right) const

{

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

}

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

{

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

}

friendbooloperator!= (constStudent&left, constStudent&right); //дружественнаяфункция, нонеметодкласса

constchar\* getName()

{

name = 0; //могупоменятьимя

return name;

}

constchar\* getName() const//сообщаемкомпиляторуоконстантностиметода const после, а const до - этотипворвращзначения

{

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

booloperator!=(constStudent&left, constStudent&right) //мбиметодомвнутрикласса

{

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

}

std::ostream&operator<<(std::ostream&os, constStudent&student) //толькофункцияглоб

{

os<<student.getName();

returnos;

}

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

{

char buffer[200];

is>> buffer;

//что-тоделаем

returnis;

}

classGroup

{

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 = newStudent \* [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;

}

friendStudent; //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(Students)

{

}

int main()

{

usingnamespace 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 delete 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

}

#include<iostream>

using namespace std;

class MyString

{

public:

    MyString()

    {

        this->str = nullptr;

    }

    MyString(const char \*str)

    {

        int lenght = strlen(str);

        this->str = new char[lenght + 1];

        for (int i(0); i < lenght; i++)

        {

            this->str[i] = str[i];

        }

        this->str[lenght] = '**\0**';

    }

    MyString(const MyString &other)

    {

        int lenght = strlen(other.str);

        this->str = new char[lenght + 1];

        for (int i(0); i < lenght; i++)

        {

            this->str[i] = other.str[i];

        }

        this->str[lenght] = '**\0**';

    }

    ~MyString()

    {

        delete[] str;

    }

    MyString &operator=(const MyString &other)

    {

        if (this->str != nullptr)

        {

            delete[] str;

        }

        int lenght = strlen(other.str); остановка происходит в этом месте

        this->str = new char[lenght + 1];

        for (int i(0); i < lenght; i++)

        {

            this->str[i] = other.str[i];

        }

        this->str[lenght] = '**\0**';

        return \*this;

    }

    MyString& operator+(const MyString &other)

    {

        int thisLenght = strlen(this->str);

        int otherLeght = strlen(other.str);

        MyString newStr;

        newStr.str = new char[thisLenght + otherLeght + 1];

        int i = 0;

        for (; i < thisLenght; i++)

        {

            newStr.str[i] = this->str[i];

        }

        for (int j(0); j < otherLeght; j++, i++)

        {

            newStr.str[i] = other.str[i];

        }

        newStr.str[thisLenght + otherLeght] = '**\0**';

        return newStr;

    }

    void Print()

    {

        cout << str << endl;

    }

private:

    char \*str;

};

int main()

{

    MyString str("C++");

    MyString str2("Test");

    MyString result;

    result = str + str2;

    result.Print();

}