Функтор

Описать класс "студент" с полями: имя, фамилия, курс. Переопределить у этого класса оператор вывода в поток. Написать функцию заполнения вектора из класса "студент" произвольными данными. Написать функцию печати содержимого вектора. Отсортировать вектор по именам студентов по возрастанию. Поставить в первые три элемента вектора студентов самых младших курсов по возрастанию. После каждой операции выводить список студентов в выходной поток.

П.С. Для генерации студентов использовать алгоритмы.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include<Windows.h>

#include<string>

#include<list>

#include<vector>

#include<map>

#include"time.h"

#include<algorithm>

#include<iterator>

using namespace std;

class Student

{

string name;

string lastname;

int course;

public:

Student()

{

}

friend ostream& operator<<(ostream& os, const Student& s);

string& Get\_Name()

{

return name;

}

string& Get\_Lastname()

{

return lastname;

}

int& Get\_Course()

{

return course;

}

Student& operator=(const Student& s)

{

if (this == &s)

return \*this;

this->name = s.name;

this->course = s.course;

this->lastname = s.lastname;

}

bool operator==(const Student& obj)

{

if (this->course == obj.course && this->name == obj.name && this->lastname == obj.lastname)

return true;

return false;

}

};

ostream& operator<<(ostream& os, const Student& s)

{

os << s.lastname << " " << s.name << " " << s.course << " курс\n";

return os;

}

class GenerataRandomStudent

{

public:

Student operator()()

{

int k = 1;

Student s;

k = rand() % 6 + 1;

switch (k)

{

case 1:

s.Get\_Lastname() = "Lashch";

break;

case 2:

s.Get\_Lastname() = "Zhylko";

break;

case 3:

s.Get\_Lastname() = "Pasevych";

break;

case 4:

s.Get\_Lastname() = "Shvaiko";

break;

case 5:

s.Get\_Lastname() = "Zimych";

break;

case 6:

s.Get\_Lastname() = "Baran";

break;

default:

s.Get\_Lastname() = "Baran";

}

k = rand() % 6 + 1;

switch (k)

{

case 1:

s.Get\_Name() = "Anton";

break;

case 2:

s.Get\_Name() = "Vladislav";

break;

case 3:

s.Get\_Name() = "Andriy";

break;

case 4:

s.Get\_Name() = "Fedia";

break;

case 5:

s.Get\_Name() = "Kolia";

break;

case 6:

s.Get\_Name() = "Sasha";

break;

default:

s.Get\_Name() = "Sasha";

}

k = rand() % 4 + 1;

switch (k)

{

case 1:

s.Get\_Course() = 1;

break;

case 2:

s.Get\_Course() = 2;

break;

case 3:

s.Get\_Course() = 3;

break;

case 4:

s.Get\_Course() = 4;

break;

default:

s.Get\_Course() = 4;

}

return s;

}

};

class SortByName

{

public:

bool operator()(Student& left, Student& right)

{

return left.Get\_Name() < right.Get\_Name();

}

};

class SortByCourse

{

public:

bool operator()(Student& left, Student& right)

{

return left.Get\_Course() < right.Get\_Course();

}

};

class Group

{

vector<Student> group;

public:

void Show()

{

copy(group.begin(), group.end(), ostream\_iterator<Student>(cout, ""));

cout << "------------------------------------------\n";

}

void Test()

{

group.resize(rand()%10+4);

generate(group.begin(), group.end(), GenerataRandomStudent());

cout << "\t\tRandom Students\n";

Show();

sort(group.begin(), group.end(), SortByName());

cout << "\t\tSort by name\n";

Show();

vector<Student>tmp;

tmp = group;

sort(tmp.begin(), tmp.end(), SortByCourse());

for (auto it = tmp.begin(); it != tmp.begin() + 3; it++)

{

const Student a= \*it;

remove(group.begin(), group.end(), a);

}

group.erase(group.end() - 3, group.end());

group.insert(group.begin(), tmp.begin(), tmp.begin() + 3);

cout << "\tTop 3 young course to the beginning\n";

Show();

}

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(0));

Group a;

a.Test();

system("pause");

return 0;

}