#include <iostream>

#include <string>

#include <fstream>

#include <sstream>

#include <iterator>

#include <vector>

#include <algorithm>

#include <functional>

#include <time.h>

using namespace std;

struct Artefact

{

string name;

int power;

};

istream& operator >>(istream& in, Artefact& obj)

{

string temp;

getline(in, temp, ':');

obj.name = temp;

getline(in, temp, ':');

obj.power = atoi(temp.c\_str());

return in;

}

ostream& operator <<(ostream& out, const Artefact& obj)

{

out << obj.name << " " << obj.power;

return out;

}

class Monstr {

string name;

int health;

int armour;

int damage;

vector<Artefact> art;

public:

Monstr() :name("Monstr 1"), health(100), armour(0), damage(30) {}

Monstr(string n, int h, int a, int d) :name(n), health(h), armour(a), damage(d) {

if (health > 100)health = 100;

if (armour > 100)armour = 100;

if (damage > 50)damage = 50;

}

Monstr(const Monstr& obj) :name(obj.name), health(obj.health), armour(obj.armour), damage(obj.damage), art(obj.art) {}

void print(ostream& out)const {

out << "name: " << name << " health: " << health << " armour: " << armour << " damage: " << damage << "Monstr is " << alive() << " ";

copy(art.begin(), art.end(), ostream\_iterator<Artefact>(out, " "));

}

bool operator < (const Monstr& arg2) const

{

return health < arg2.health;

}

string getName()const;

void setName(string n);

int getHealth()const;

void setHealth(int h);

int getArmour()const;

void setArmour(int h);

int getDamage()const;

void setDamage(int h);

bool alive()const;

void healing(int health\_plus);

void attack(Monstr& obj)const;

friend istream& operator >>(istream& in, Monstr& obj);

bool greater\_80(const Monstr& obj);

};

string Monstr::getName()const { return name; }

void Monstr::setName(string n) { name = n; }

int Monstr::getHealth()const { return health; }

void Monstr::setHealth(int h) { health = (h > 100) ? 100 : h; }

int Monstr::getArmour()const { return armour; }

void Monstr::setArmour(int a) { armour = (a > 100) ? 100 : a; }

int Monstr::getDamage()const { return damage; }

void Monstr::setDamage(int d) { damage = (d > 50) ? 50 : d; }

istream& operator >>(istream& in, Monstr& obj)

{

string temp;

getline(in, temp, ';');

obj.setName(temp);

getline(in, temp, ';');

obj.setHealth(atoi(temp.c\_str()));

getline(in, temp, ';');

obj.setArmour(atoi(temp.c\_str()));

getline(in, temp, ';');

obj.setDamage(atoi(temp.c\_str()));

getline(in, temp);

stringstream s(temp);

vector<Artefact> a;

copy(istream\_iterator<Artefact>(s), istream\_iterator<Artefact>(), back\_inserter(a));

obj.art = a;

return in;

}

ostream& operator <<(ostream& out, const Monstr& obj)

{

obj.print(out);

return out;

}

bool Monstr::alive()const

{

return (health > 0);

}

void Monstr::healing(int health\_plus)

{

setHealth(health + health\_plus);

}

void Monstr::attack(Monstr& obj) const

{

if (!alive()) return;

if (this == &obj) return;

int delta = obj.armour - damage;

if (delta < 0)

{

obj.armour = 0;

obj.health += delta;

}

else

{

obj.armour = delta;

}

}

bool cmp\_monstr(const Monstr& obj1, const Monstr& obj2)

{

return obj1.getDamage() < obj2.getDamage();

}

bool greater\_80(const Monstr& obj)

{

return obj.getHealth() > 80;

}

bool less\_50(const Monstr& obj)

{

return obj.getHealth() < 50;

}

struct cmp\_x

{

int x\_;

cmp\_x(int x) :x\_(x) {}

bool operator () (const Monstr& obj)

{

return obj.getHealth() < x\_;

}

};

int main() {

ifstream fin("Input.txt");

if (!fin)

{

cout << "File not found";

return 1;

}

vector<Monstr> monstrs;

copy(istream\_iterator<Monstr>(fin), istream\_iterator<Monstr>(), back\_inserter(monstrs));

copy(monstrs.begin(), monstrs.end(), ostream\_iterator<Monstr>(cout, "\n"));

sort(monstrs.begin(), monstrs.end(), cmp\_monstr);

cout << endl;

copy(monstrs.begin(), monstrs.end(), ostream\_iterator<Monstr>(cout, "\n"));

sort(monstrs.begin(), monstrs.end(), [](const Monstr& obj1, const Monstr& obj2) {return obj1.getName() < obj2.getName(); });

cout << endl;

copy(monstrs.begin(), monstrs.end(), ostream\_iterator<Monstr>(cout, "\n"));

vector<Monstr>monstrs2;

vector<Monstr >::iterator it;

it = find\_if(monstrs.begin(), monstrs.end(), greater\_80);

copy(it, monstrs.end(), back\_inserter(monstrs2));

cout << endl;

copy(monstrs2.begin(), monstrs2.end(), ostream\_iterator<Monstr>(cout, "\n"));

vector<string>names;

vector<int>healthes;

transform(monstrs.begin(), monstrs.end(), back\_inserter(names), mem\_fun\_ref(&Monstr::getName));

copy(names.begin(), names.end(), ostream\_iterator<string>(cout, " "));

transform(monstrs.begin(), monstrs.end(), back\_inserter(healthes), mem\_fun\_ref(&Monstr::getHealth));

cout << endl;

copy(healthes.begin(), healthes.end(), ostream\_iterator<int>(cout, " "));

int count\_live = count\_if(monstrs.begin(), monstrs.end(), mem\_fun\_ref(&Monstr::alive));

cout << "Alive: " << count\_live << endl;

int count\_health = count\_if(monstrs.begin(), monstrs.end(), less\_50);

cout << "Count health: " << count\_health << endl;

Monstr monstr1("", 50, 57, 89);

count\_health = count\_if(monstrs.begin(), monstrs.end(), bind2nd(less<Monstr>(), monstr1));

cout << "Count health2: " << count\_health << endl;

int x = 50;

count\_health = count\_if(monstrs.begin(), monstrs.end(), [x](const auto& d) {return d.getHealth() < x; });

cout << "Count health3: " << count\_health << endl;

count\_health = count\_if(monstrs.begin(), monstrs.end(), cmp\_x(70));

cout << "Count health4: " << count\_health << endl;

return 0;

}

#include <iostream>

#include <fstream>

#include <iterator>

#include <algorithm>

#include <set>

#include <map>

using namespace std;

ostream& operator << (ostream& out, const map<int, int>::value\_type& p)

{

out << p.first << " " << p.second;

return out;

}

int main() {

ifstream fin("Input 2.txt");

if (!fin)

{

cout << "File not found";

return 1;

}

set<string> Exceptions;

while (!fin.eof())

{

string temp;

fin >> temp;

Exceptions.emplace(temp);

}

copy(Exceptions.begin(), Exceptions.end(), ostream\_iterator<string>(cout, " "));

fin.close();

cout << endl;

fin.open("Input.txt");

if (!fin)

{

cout << "File not found";

return 1;

}

map <int, int> Words;

while (!fin.eof())

{

string temp;

fin >> temp;

if (Exceptions.count(temp) != 0)

continue;

Words[temp.size()]++;

}

for\_each(Words.begin(), Words.end(), [](const map<int, int>::value\_type& obj) {cout << obj.first << " " << obj.second << endl; });

//copy(m.begin(), m.end(), ostream\_iterator< const Mpoint::value\_type&>(cout, "\n")

}