**Задание**

* 1. ***Условие задачи***

**Комплектующие PC**. Определить иерархию комплектующих PC. Создать объекты комплектующих с различными характеристиками. Собрать из комплектующих готовые PC. Найти PC c наибольшей частотой.

* 1. ***Алгоритм выполнения программы:***

Создаем класс, который включает в себя комплектующие ПК: процессор, видеокарта, карта памяти, которые также являются классами со своими переменными. Считываем информацию о ПК из файла в вектор. Сравниваем частоту процессоров пк и находим наибольший.

Код программы отображен в листинге 1.

* 1. ***Листинг 1 – Программа «Комплектующие* PC*»***

#include <iostream>

using namespace std;

class basic {

protected:

string brand;

string name;

double cost;

public:

basic(string b,string n, double c) {

brand = b;

name = n;

cost = c;

}

basic() {

brand = "";

name = "";

cost = 0;

}

string getBrand() {

return brand;

}

string getName() {

return name;

}

double getCost() {

return cost;

}

};

class proccesor : public basic {

int cores;

double frequency;

public:

proccesor(string b, string n, int cor, double f, double cos) {

brand = b;

name = n;

cost = cos;

cores = cor;

frequency = f;

}

proccesor() {

brand = "";

name = "";

cost = 0;

cores = 0;

frequency = 0;

}

int getCores() {

return cores;

}

double getFrequency() {

return frequency;

}

friend ostream& operator<<(ostream& out, proccesor pc) {

out << pc.brand << " " << pc.name << " " << pc.cores << " cores " << pc.frequency << "MHz " << pc.cost << "$" << endl;

return out;

}

};

class videocard : public basic {

double memory;

int type;

public:

videocard(string b, string n, double m, int t, double cos) {

brand = b;

name = n;

cost = cos;

memory = m;

type = t;

}

videocard() {

brand = "";

name = "";

cost = 0;

memory = 0;

type = 0;

}

double getMemory() {

return memory;

}

int getType() {

return type;

}

friend ostream& operator<<(ostream& out, videocard pc) {

out << pc.brand << " " << pc.name << " " << pc.memory << "Gb " << pc.type << " bit " << pc.cost << "$" << endl;

return out;

}

};

class Mem : public basic {

double memory;

double frequency;

public:

Mem(string b, string n, double m, double f, double cos) {

brand = b;

name = n;

cost = cos;

memory = m;

frequency = f;

}

Mem() {

brand = "";

name = "";

cost = 0;

memory = 0;

frequency = 0;

}

double getMemory() {

return memory;

}

double getFrequency() {

return frequency;

}

friend ostream& operator<<(ostream& out, Mem pc) {

out << pc.brand << " " << pc.name << " " << pc.memory << "Gb " << pc.frequency << "MHz " << pc.cost << "$" << endl;

return out;

}

};

class PC {

proccesor proc;

videocard card;

Mem mem;

double cost;

public:

PC(proccesor p, videocard c, Mem m) {

proc = p;

card = c;

mem = m;

cost = p.getCost() + c.getCost() + m.getCost();

}

PC() {

proccesor p;

videocard c;

Mem m;

proc = p;

card = c;

mem = m;

cost = p.getCost() + c.getCost() + m.getCost();

}

proccesor getCPU() {

return proc;

}

videocard getVideocard() {

return card;

}

Mem getMemecard() {

return mem;

}

double getCost() {

return cost;

}

friend ostream& operator<<(ostream& out, PC pc) {

out << "CPU " << pc.proc << "Video card " << pc.card << "Memory card " << pc.mem << "Cost " << pc.cost << "$" << endl;

return out;

}

};

#include <iostream>

#include <vector>

#include <fstream>

#include "Components.h"

using namespace std;

vector<PC> initialize() {

ifstream file("Text.txt");

vector<PC> computers;

string brand = "", name = "";

double cost = 0, frequency = 0, memory = 0;

int type = 0, cores = 0;

while (!file.eof()) {

file >> brand >> name >> cores >> frequency >> cost;

proccesor proc(brand, name, cores, frequency, cost);

file >> brand >> name >> memory >> type >> cost;

videocard card(brand, name, memory, type, cost);

file >> brand >> name >> memory >> frequency >> cost;

Mem mem(brand, name, memory, frequency, cost);

PC pc(proc, card, mem);

computers.push\_back(pc);

}

return computers;

}

int main() {

vector<PC> computers = initialize();

vector<PC> ::iterator iter = computers.begin();

int i = 0, maxi = 0;;

double maxf = computers[i].getCPU().getFrequency();

while (iter!=computers.end())

{

if (computers[i].getCPU().getFrequency() > maxf) {

maxf = computers[i].getCPU().getFrequency();

maxi = i;

}

cout << "\nPC\n";

cout << \*iter;

iter++;

i++;

}

cout << "\n\n\nPC with highest CPU frequency\n" << computers[maxi];

}

* 1. ***Контрольный тест:***

1)

