МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Чувашский государственный университет имени И. Н. Ульянова»

Факультет информатики и вычислительной техники

Кафедра математического и аппаратного обеспечения

информационных систем

Расчетно-графическая работа по дисциплине «Языки программирования»

на тему:

«Проектирование и реализация класса «Рабочие фабрики 37»

Выполнил:

Студент группы ИВТ-21-22

Арманов Кирилл Константинович

Проверил:

Ст. преподаватель Ильина Л.А.

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Чебоксары 2022

# **Содержание**

[**Содержание** 2](#_Toc135153034)

[**Введение** 3](#_Toc135153035)

[**Постановка задачи** 5](#_Toc135153036)

[**Теоретическая часть** 6](#_Toc135153037)

[**Практическая часть** 7](#_Toc135153038)

[Создаю класс «Point» 7](#_Toc135153039)

[**Содержание программы** 12](#_Toc135153040)

[**Список использованной литературы** 21](#_Toc135153041)

[Приложение 1 22](#_Toc135153042)

[Приложение 2 27](#_Toc135153043)

[Примечание 27](#_Toc135153044)

# **Введение**

Объектно-ориентированное программирование (сокр. ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Идеологически ООП — подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости, что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при реализации крупных проектов. Управляемость для иерархических систем предполагает минимизацию избыточности данных (аналогичную нормализации) и их целостность, поэтому созданное удобно управляемым — будет и удобно пониматься. Таким образом, через тактическую задачу управляемости решается стратегическая задача — транслировать понимание задачи программистом в наиболее удобную для дальнейшего использования форму.

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью:

Абстракция для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счёте — контекстное понимание предмета, формализуемое в виде класса; инкапсуляция для быстрой и безопасной организации собственно иерархической управляемости: чтобы было достаточно простой команды «что делать», без одновременного уточнения как именно делать, так как это уже другой уровень управления;

Наследование для быстрой и безопасной организации родственных понятий: чтобы было достаточно на каждом иерархическом шаге учитывать только изменения, не дублируя всё остальное, учтенное на предыдущих шагах; полиморфизм для определения точки, в которой единое управление лучше распараллелить или наоборот — собрать воедино.

Целью расчетно-графической работы (РГР) является закрепление практических навыков самостоятельной работы и отладки любой сложности программного изделия с применением современных навыков и методов программирования; овладение современными методами наглядного представления результатов решения, организации диалога с ЭВМ

# 

# **Постановка задачи**

Разработка и реализация класса содержащий сведения о рабочих предприятия: ФИО, номер цеха, количество продукции А, В, С, зарплата сотрудника

Написать программу, позволяющую получить следующую информацию: общее количество изделий категорий А, В, С, собранных рабочим цеха X, ведомость заработной платы рабочих цеха X, средний размер заработной платы работников этого цеха. Данные и результаты должны храниться в файлах.

Предусмотреть возможность добавления информации, удаления отдельных записей, сортировки по заданному полю.

# **Теоретическая часть**

Создаем класс, содержащий сведения о сотрудниках института. В модификатор доступа private создаем переменные: name(имя), locate(номер цеха), a,b,c(продукция A,B,C соответсвенно), num(номер сотрудника), salary(зарплата).

В открытой части класса “public” создаем функции для получения значений и их редактирования.

Сделать возможность добавления информации:

* создаем объект типа класс
* введем данные из консоли
* считываем объект в файл

Сотрудник добавлен

Удаления отдельных записей:

* считываем вектор в файл (упуская объект с совпадающим номером)

Сотрудник удален.

Считывание данных с файла.

Вывод в файл. На выводе использую поток fstream c с режимом создания файла и добавлением в конец.

# **Практическая часть**

## Создаю класс «Point»

В модификатор доступа private создаю переменные:

Переменные «name, locate, a, b, c» хранят информацию о фамилии, имени, отчестве, названии департамента и продукции.

Переменныеs «alary, num» хранят информацию о зарплате и номере сотрудника

Статическая переменная int temp объявлена без инициализации значения в какой-либо функции. Она будет существовать в памяти на протяжении жизни программы и будет общей для всех объектов класса, т.е. ее значение будет совпадать для всех экземпляров класса. Данная переменная может использоваться для отслеживания количества созданных объектов данного класса.

В модификатор доступа public создаю методы:

class Point  
{  
    private:  
        string name, locate;  
        int a,b,c,num,salary;  
    public:  
        void put(Point p, vector<Point>& worker)  
        {  
            ifstream fin;  
            fin.open("text32-1.txt");  
            if (!fin)    cout << "File not founded!!";  
              
            while(!fin.eof())  
            {  
                fin >> p.name >> p.locate >> p.a >> p.b >> p.c;  
                p.salary = p.a\*5 + p.b\*15 + p.c\*20;  
                p.num = temp;  
                temp++;  
                worker.push\_back(p);  
            }  
            fin.close();  
        }  
          
        void show(vector<Point>& worker)  
        {  
            int n = 0;  
            cout << "\n     |WORKERS DATA|\n"  
                << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  
            while(n < worker.size())  
            {  
                cout << worker[n].name << " " << worker[n].locate  
                    << "  " << worker[n].a << " " << worker[n].b  
                    << " " << worker[n].c << "  " << worker[n].salary  
                    << "$" << endl;  
                n++;  
            }  
            cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n";  
        }      
          
        void push(Point p, vector<Point>& worker)  
        {  
            ofstream fout;  
            fout.open("text32-2.txt");  
            if (!fout) cout << "File not founded!!";  
              
            int n,d1,d2,d3;  
            double a1,a2,a3;  
            n = a1 = a2 = a3 = d1 = d2 = d3 = 0;  
              
            while(n < worker.size())  
            {      
                fout << worker[n].num << ") " <<worker[n].name << " " << worker[n].locate << " -- "  
                    << worker[n].a << " " << worker[n].b << " "  
                    << worker[n].c << " --  " << worker[n].salary << "$" << endl;  
                  
                if(worker[n].locate == "A1")  
                {  
                    a1 = a1 + worker[n].salary;  
                    d1 += 1;  
                }  
                      
                if(worker[n].locate == "A2")  
                    {  
                        a2 = a2 + worker[n].salary;  
                        d2 += 1;  
                    }  
                      
                if(worker[n].locate == "A3")  
                    {  
                        a3 = a3 + worker[n].salary;  
                        d3 += 1;  
                    }  
                n++;  
            }  
            fout << endl << "Workshop salary:\nA1 -- " << a1/d1  
                << "$\nA2 -- " << a2/d2 << "$\nA3 -- " << a3/d3 << "$";  
            fout.close();  
            cout << "Saved successfully!\n\n";  
        }  
          
        void del(vector<Point>& worker)  
        {  
            cout << "Enter number of worker to delete: ";  
            int n;    bool found = false;  
            cin >> n;  
            for(auto it = worker.begin(); it != worker.end();)  
            {  
                if(it->num == n)  
                {  
                    it = worker.erase(it);  
                    found = true;  
                    break;  
                }  
                it++;  
            }  
            cout << endl;  
        }  
              
        void enter(vector<Point>& worker)  
        {  
            Point t;  
            int a,b,c; a = b = c = 0;  
            int num = temp;    temp++;  
            string name,locate; name = locate = "";  
              
            cout << "Enter candidate information:\nName: ";  
                cin >> t.name;  
                  
                int k = 1;  
                while(k != 0)  
                {  
                    for(auto it = worker.begin(); it < worker.end();)  
                        {  
                            if(t.name == it->name)  
                            {  
                                cout << "Such worker already exists. Enter name again: ";  
                                cin >> t.name;  
                            }  
                            else ++it;  
                        }  
                    k = 0;  
                }  
                  
            cout << "Number workshop: ";  
                cin >> t.locate;  
            cout << "Quantity products -A- : ";  
                cin >> t.a;  
            cout << "Quantity products -B- : ";  
                cin >> t.b;  
            cout << "Quantity products -C- : ";  
                cin >> t.c;    cout << "\n";  
                t.salary = t.salary = t.a\*5 + t.b\*15 + t.c\*20;  
            worker.push\_back(t);  
        }  
          
        void sorting(vector<Point>& worker)  
        {  
            int n = -1;  
            while(n)  
            {  
                switch(n)  
                {  
                    case 1:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.name < n2.name;});  
                        n = 0;  
                        break;  
                    case 2:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.locate < n2.locate;});  
                        n = 0;  
                        break;  
                    case 3:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.a < n2.a;});  
                        n = 0;  
                        break;  
                    case 4:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.b < n2.b;});  
                        n = 0;  
                        break;  
                    case 5:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.c < n2.a;});  
                        n = 0;  
                        break;          
                    case 6:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.salary < n2.salary;});  
                        n = 0;  
                        break;  
                    case 7:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.num < n2.num;});  
                        n = 0;  
                        break;  
                    default:  
                        {  
                            cout << "Sort by:\n1 - Name\n2 - Workshop\n3 - Products -A- "  
                                <<"\n4 - Products -B- \n5 - Products -C- \n6 - Salary"  
                                <<"\n7 - Worker number\nChoosen: ";  
                            cin >> n; cout << "\n";  
                            if(n < 0 or n > 7)  
                            {  
                                cout << "\nDenied! Enter choosen again\n";  
                                cin >> n;  
                            }  
                            break;      
                        }  
                }  
            }  
        }  
};

Выше приведен код определения класса `Point`.

# **Содержание программы**

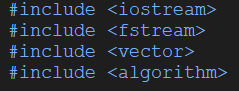


Рис. 1 Список подключаемых бибиотек.

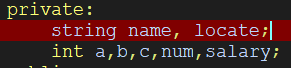


Рис. 2 Переменные класса, хранимые в закрытом доступе.

Функция «put» (рис. 3) для считывания данных о рабочих завода. На входе принимает обьект класса для работы с закрытыми переменными и вектор хранящий данные о сотруднике(далее «вектор Х»).

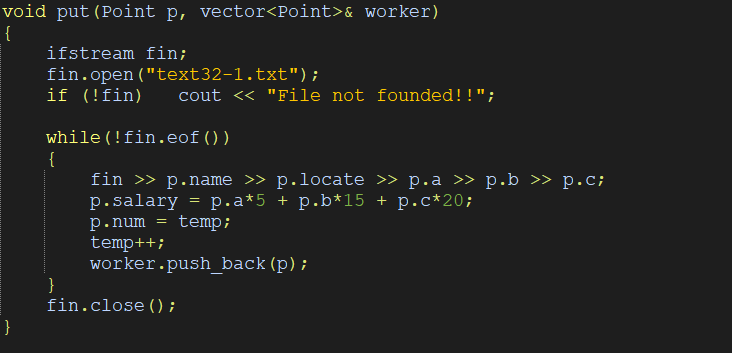


Рис. 3 Ввод данных в контейнер для хранения информации.

Функция «show» (рис. 4) для просмотра информации о рабочих завода. На вход получает вектор Х. Параметр «n < workers.size()» в цикле «while» работает, пока в векторе есть информация о рабочих. Пример работы (рис. 5)

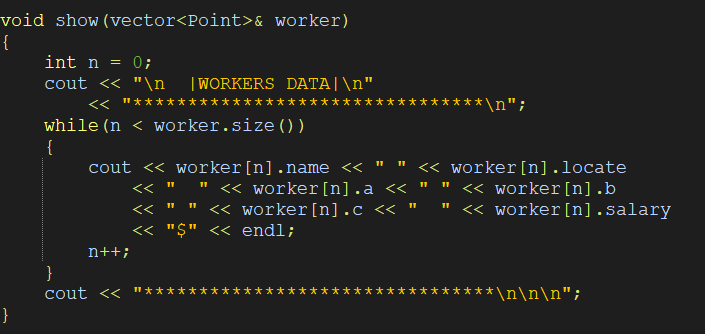


Рис. 4 Функция для просмотра данных о рабочих.

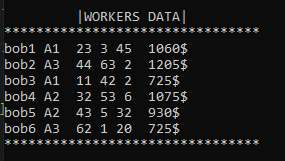


Рис. 5 Список сотрудников мастерской и их данные.

Функция «push» (рис. 6) служит для сохранения информации в файл. На входе принимает обьект класса для работы с закрытыми переменными и вектор Х. В функции с помощью переменной «fout» получаем доступ к файлу. Переменные «d1,d2,d3» созданы для сравнения цехов рабочих. Переменные «a1,a2,a3» хранят данные о средней зарплате цеха.

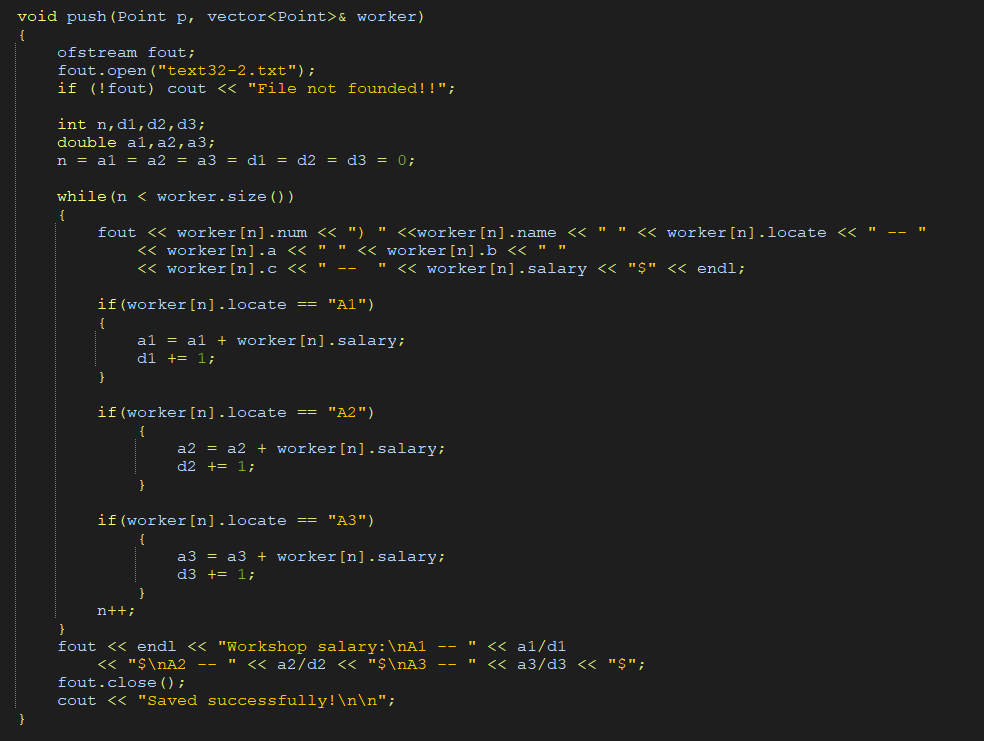


Рис. 6 Сохранение данных в файле для хранения информации.

Функция «del» (рис. 7) служит для удаления сотрудника. На вход получает вектор Х. Проверкой служит цикл, который перебирает данные вектора посредством итераторов. Переменная «found» хранит в себе 1 или 0, для удостоверения успешно выполненной функции.

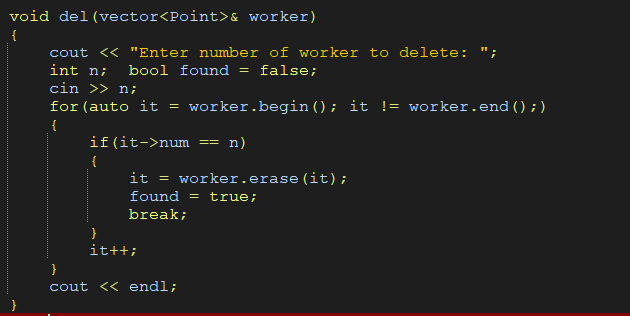


Рис. 7 Удаление данных сотрудника.

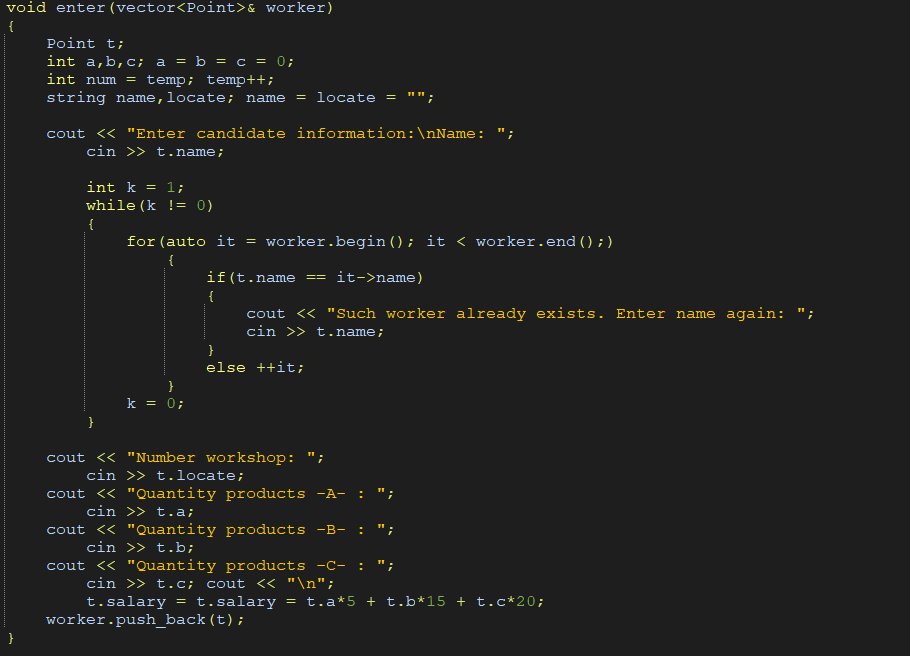
Функция «enter» (рис. 8) служит для ввода данных для нового сотрудника цеха. На вход получает вектор Х. Здесь Поочередно вводятся ФИО, цех, продукция А,Б,С , после вычисляется зарплата данного сотрудника. После все эти данные добавляются в конец вектора Х до их сохранения в файл.

Рис. 8 Добавление нового сотрудника базу данных.

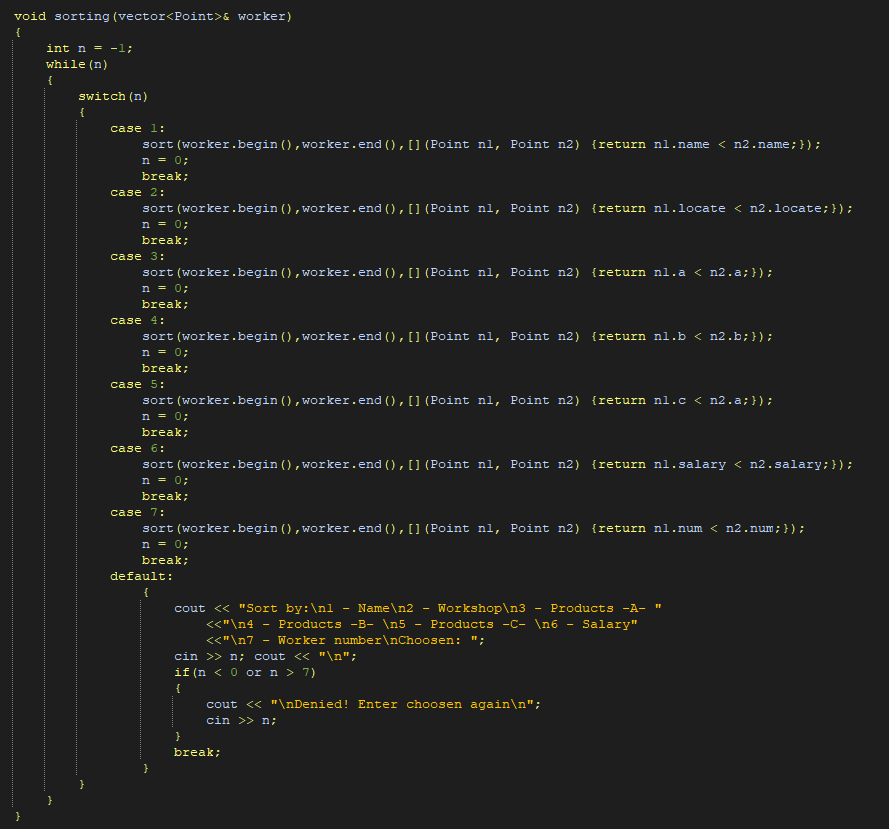
Функция «sorting» (рис. 9) служит для сортировки данных по различным параметрам. На вход получает вектор Х. В ней представлен метод «switch» для создания меню сортировки. Все данных сохраняются в вектор.

Рис. 9 Сортировка списка сотрудников по выбранному параметру (рис. 10)

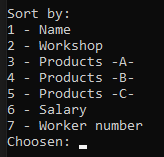


Рис. 10 Выбор параметра для сортировки списка.

В главной функции «main» (рис. 11) создается классовый объект «р» для работы с классом и его методами, вектор объектов «worker» для хранения информации рабочих, функция «put» начинает работу с файлом и классом сохраняя данные в вектор для дальнейшего пользования. Также методом «switch» реализуется меню с выбором действий для удобства (рис.12).

Рис. 11 Главный пункт управления программой.

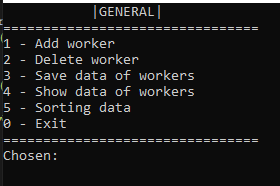


Рис. 12 Главное меню выбора действий.

**Заключение**

Таким образом, через тактическую задачу управляемости разрешил задачу — транслировать понимание задачи программистом в наиболее удобную для дальнейшего использования форму.

Основные принципы структурирования в случае ООП связаны с различными аспектами базового понимания предметной задачи, которое требуется для оптимального управления соответствующей моделью.

В расчетно-графической работе (РГР) закрепил практические навыки самостоятельной работы и отладки любой сложности программного изделия с применением современных навыков и методов программирования; овладение современными методами наглядного представления результатов решения, организации диалога с ЭВМ.

# **Список использованной литературы**

1. Павловская Т. А. C/C++. Программирование на языке высокого уровня: [учебник для вузов по направлению "Информатика и вычислительная техника"] / Павловская Т. А. – СПб. и др.: Питер, 2010. – 460с.

2. Лаптев В. В. C++. Объектно-ориентированное программирование: задачи и упражнения: [учебное пособие для вузов по направлению "Информатика и вычислительная техника"] / Лаптев В. В., Бокова А. В., Морозов А. В. – СПб.: Питер, 2008. – 287с.

# Приложение 1

#include <iostream>  
#include <fstream>  
#include <vector>  
#include <algorithm>

using namespace std;

int temp = 1;  
class Point  
{  
    private:  
        string name, locate;  
        int a,b,c,num,salary;  
    public:  
        void put(Point p, vector<Point>& worker)  
        {  
            ifstream fin;  
            fin.open("text32-1.txt");  
            if (!fin)    cout << "File not founded!!";  
              
            while(!fin.eof())  
            {  
                fin >> p.name >> p.locate >> p.a >> p.b >> p.c;  
                p.salary = p.a\*5 + p.b\*15 + p.c\*20;  
                p.num = temp;  
                temp++;  
                worker.push\_back(p);  
            }  
            fin.close();  
        }  
          
        void show(vector<Point>& worker)  
        {  
            int n = 0;  
            cout << "\n     |WORKERS DATA|\n"  
                << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  
            while(n < worker.size())  
            {  
                cout << worker[n].name << " " << worker[n].locate  
                    << "  " << worker[n].a << " " << worker[n].b  
                    << " " << worker[n].c << "  " << worker[n].salary  
                    << "$" << endl;  
                n++;  
            }  
            cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n";  
        }      
          
        void push(Point p, vector<Point>& worker)  
        {  
            ofstream fout;  
            fout.open("text32-2.txt");  
            if (!fout) cout << "File not founded!!";  
              
            int n,d1,d2,d3;  
            double a1,a2,a3;  
            n = a1 = a2 = a3 = d1 = d2 = d3 = 0;  
              
            while(n < worker.size())  
            {      
                fout << worker[n].num << ") " <<worker[n].name << " " << worker[n].locate << " -- "  
                    << worker[n].a << " " << worker[n].b << " "  
                    << worker[n].c << " --  " << worker[n].salary << "$" << endl;  
                  
                if(worker[n].locate == "A1")  
                {  
                    a1 = a1 + worker[n].salary;  
                    d1 += 1;  
                }  
                      
                if(worker[n].locate == "A2")  
                    {  
                        a2 = a2 + worker[n].salary;  
                        d2 += 1;  
                    }  
                      
                if(worker[n].locate == "A3")  
                    {  
                        a3 = a3 + worker[n].salary;  
                        d3 += 1;  
                    }  
                n++;  
            }  
            fout << endl << "Workshop salary:\nA1 -- " << a1/d1  
                << "$\nA2 -- " << a2/d2 << "$\nA3 -- " << a3/d3 << "$";  
            fout.close();  
            cout << "Saved successfully!\n\n";  
        }  
          
        void del(vector<Point>& worker)  
        {  
            cout << "Enter number of worker to delete: ";  
            int n;    bool found = false;  
            cin >> n;  
            for(auto it = worker.begin(); it != worker.end();)  
            {  
                if(it->num == n)  
                {  
                    it = worker.erase(it);  
                    found = true;  
                    break;  
                }  
                it++;  
            }  
            cout << endl;  
        }  
              
        void enter(vector<Point>& worker)  
        {  
            Point t;  
            int a,b,c; a = b = c = 0;  
            int num = temp;    temp++;  
            string name,locate; name = locate = "";  
              
            cout << "Enter candidate information:\nName: ";  
                cin >> t.name;  
                  
                int k = 1;  
                while(k != 0)  
                {  
                    for(auto it = worker.begin(); it < worker.end();)  
                        {  
                            if(t.name == it->name)  
                            {  
                                cout << "Such worker already exists. Enter name again: ";  
                                cin >> t.name;  
                            }  
                            else ++it;  
                        }  
                    k = 0;  
                }  
                  
            cout << "Number workshop: ";  
                cin >> t.locate;  
            cout << "Quantity products -A- : ";  
                cin >> t.a;  
            cout << "Quantity products -B- : ";  
                cin >> t.b;  
            cout << "Quantity products -C- : ";  
                cin >> t.c;    cout << "\n";  
                t.salary = t.salary = t.a\*5 + t.b\*15 + t.c\*20;  
            worker.push\_back(t);  
        }  
          
        void sorting(vector<Point>& worker)  
        {  
            int n = -1;  
            while(n)  
            {  
                switch(n)  
                {  
                    case 1:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.name < n2.name;});  
                        n = 0;  
                        break;  
                    case 2:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.locate < n2.locate;});  
                        n = 0;  
                        break;  
                    case 3:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.a < n2.a;});  
                        n = 0;  
                        break;  
                    case 4:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.b < n2.b;});  
                        n = 0;  
                        break;  
                    case 5:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.c < n2.a;});  
                        n = 0;  
                        break;          
                    case 6:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.salary < n2.salary;});  
                        n = 0;  
                        break;  
                    case 7:  
                        sort(worker.begin(),worker.end(),[](Point n1, Point n2) {return n1.num < n2.num;});  
                        n = 0;  
                        break;  
                    default:  
                        {  
                            cout << "Sort by:\n1 - Name\n2 - Workshop\n3 - Products -A- "  
                                <<"\n4 - Products -B- \n5 - Products -C- \n6 - Salary"  
                                <<"\n7 - Worker number\nChoosen: ";  
                            cin >> n; cout << "\n";  
                            if(n < 0 or n > 7)  
                            {  
                                cout << "\nDenied! Enter choosen again\n";  
                                cin >> n;  
                            }  
                            break;      
                        }  
                }  
            }  
        }  
};

int main()  
{  
    Point p;  
    vector<Point> worker;  
    p.put(p,worker);  
      
    int n = -1;  
    while(n)  
    {  
        switch(n)  
        {  
            case 1:  
                p.enter(worker);  
                n = -1;  
                break;  
            case 2:  
                p.del(worker);  
                n = -1;  
                break;  
            case 3:  
                p.push(p,worker);  
                n = -1;  
                break;  
            case 4:  
                p.show(worker);  
                n = -1;  
                break;  
            case 5:  
                p.sorting(worker);  
                p.show(worker);  
                n = -1;  
                break;  
            default:  
                {  
                    cout << "       |GENERAL|"  
                        <<"\n================================\n"  
                        <<"1 - Add worker\n2 - Delete worker\n3 "  
                        << "- Save data of workers\n4 - Show data of"  
                        << " workers\n5 - Sorting data\n0 - Exit\n"  
                        << "================================\nChosen: ";  
                    cin >> n; cout << "\n";  
                            if(n < 0 or n > 5)  
                            {  
                                cout << "\nDenied! Enter choosen again\n";  
                                cin >> n;  
                            }  
                            break;          
                }  
                  
        }  
    }      
}

# Приложение 2

Файл:1

bob1 A1 23 3 45

bob2 A3 44 63 2

bob3 A1 11 42 2

bob4 A2 32 53 6

bob5 A2 43 5 32

bob6 A3 62 1 20