Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Курский государственный университет»

Кафедра программного обеспечения и администрирования информационных систем

Направление подготовки математическое обеспечение и администрирование информационных систем

Форма обучения очная

Отчет

по лабораторной работе №2

«Практическая реализация классов»

Выполнил:
студент группы 213 Тихонов Е.Е.

Проверил:
старший преподаватель кафедры ПОиАИС Ураева Е.Е.

Курск, 2021

Цель работы: изучить особенности реализации классов на языке C++.

Задание

Задача 1. Определить класс «Список сотрудников». Объект класса содержит список, состоящий из нескольких сотрудников. Свойства каждого из сотрудников: ФИО, табельный номер, количество отработанных часов за месяц, почасовой тариф. Рабочее время свыше 144 часов считается сверхурочным и оплачивается в двойном размере. Обязательные методы класса:

- определение списка всех сотрудников с указанием размера заработной платы каждого за вычетом подоходного налога (12% от суммы заработка);

- определение списка всех сотрудников, отсортированного по фамилиям.

Разработка алгоритма

Задача 1

Входные данные: нет.

Выходные данные: текстовые данные в консоли.

Текст программы

Текст программы для решения задачи 1

```
#include <iostream>
#include <string>

using namespace std;

class Worker
{
    private:
```

```
string Name;
int ReportCard;
float PricePerHour;
int WorkedHours;

//property getters
public:
string getName()
{
    return this->Name;
}
int getReportCard()
{
    return this->ReportCard;
}
float getPricePerHour()
{
    return this->PricePerHour;
}
int getWorkedHours()
{
    return this->WorkedHours;
}

//property setters
private:
void setName(string Name)
{
    this->Name = Name;
}
void setReportCard(int ReportCard)
{
    this->ReportCard = ReportCard;
```

```
    }
    void setPricePerHour(float PricePerHour)
    {
        this->PricePerHour = PricePerHour;
    }
    void setWorkedHours(int WorkedHours)
    {
        this->WorkedHours = WorkedHours;
    }

    public:
     Worker(string Name, int ReportCard, float PricePerHour = 100, int
WorkedHours = 10)
    {
        setName(Name);
        setReportCard(ReportCard);
        setPricePerHour(PricePerHour);
        setWorkedHours(WorkedHours);
    }

    void Print()
    {
        cout << this->Name << " ";
        cout << this->ReportCard << " ";
        cout << to_string(this->PricePerHour) << " ";
        cout << to_string(this->WorkedHours) << endl;
    }
};

class  WorkersList
{
    private:
    Worker** _workers = nullptr;
```

```
int _workersCount = 0;
//getters
public:
int getWorkersCount()
{
    return this->_workersCount;
}


public:
WorkersList(Worker** Workers, int WorkersCount)
{
    Add(Workers, WorkersCount);
}

WorkersList()
{
    //...
}

void Add(Worker** Workers, int WorkersCount)
{
    ResizeList(_workersCount + WorkersCount);

    for (size_t i = 0; i < WorkersCount; i++)
    {
        _workers[_workersCount - WorkersCount + i] = Workers[i];
    }
}

void Add(Worker* worker)
{
    Worker** WorkerPtr = &worker;
```

```
        Add(WorkerPtr, 1);
    }

    Worker* Remove(int Indx)//exclude worker flom list
    {
        if (Indx < _workersCount && Indx >= 0)
        {
            Worker* RemovedWorker = _workers[Indx];

            _workers[Indx] = nullptr;
            ResizeList(_workersCount - 1);

            return RemovedWorker;
        }
        else
        {
            return nullptr;
        }
    }

     void Delete(int Indx)//exclude worker flom list and delete worker
from mem
    {
        if (Indx < _workersCount && Indx >= 0)
        {
            delete _workers[Indx];

            _workers[Indx] = nullptr;
            ResizeList(_workersCount - 1);
        }
    }
```

```
Worker* At(int Indx)
{
    if (Indx < _workersCount && Indx >= 0)
    {
        return _workers[Indx];
    }
    else
    {
        return nullptr;
    }
}

float GetWorkerSalary(int Indx)
{
    float Salary = 0;
    if (Indx < _workersCount)
    {
        int Hours = _workers[Indx]->getWorkedHours();
        int Price = _workers[Indx]->getPricePerHour();;
        if (Hours > 144)
        {
            Salary = Price * 144 + Price * 2 * (Hours - 144);
        }
        else
        {
            Salary = Price * Hours;
        }
    }

    return Salary - Salary * 0.12;
}

float* GetSalaries()
```

```
    {
        float* Salaries = new float[_workersCount];
        for (size_t i = 0; i < _workersCount; i++)
        {
            Salaries[i] = GetWorkerSalary(i);
        }
        return Salaries;
    }
    Worker** GetSortedWorkers()
    {
        Worker** SortedWorkers = new Worker * [_workersCount];

        for (size_t i = 0; i < _workersCount; i++)
        {
            SortedWorkers[i] = _workers[i];
        }

        Worker* tmp;
        for (size_t i = 0; i < _workersCount; i++)
        {
            for (size_t j = i + 1; j < _workersCount; j++)
            {
                                    if  (SortedWorkers[j]->getName()  <
SortedWorkers[i]->getName())
                {
                    tmp = SortedWorkers[i];
                    SortedWorkers[i] = SortedWorkers[j];
                    SortedWorkers[j] = tmp;
                }

            }
        }
```

```
        return SortedWorkers;
    }

    private:
    void ResizeList(int NewSize)
    {
        Worker** NewWorkers = new Worker * [NewSize];
          for (size_t i = 0, j = 0; i < NewSize && j < _workersCount;
j++)
        {
            if (_workers[j] != nullptr)
            {
                NewWorkers[i] = _workers[j];
                i++;
            }
        }

        if (_workers != nullptr)
            delete _workers;

        _workers = NewWorkers;
        _workersCount = NewSize;
    }

    public:
    Worker* operator [] (unsigned index)
    {
        if (index >= 0 && index < _workersCount)
            return At(index);
        else
            throw std::out_of_range("error");
    }
};
```

```
WorkersList operator +(WorkersList& ob1, WorkersList& ob2)
{
    WorkersList mergeList;

    int count1 = ob1.getWorkersCount();
    for (size_t i = 0; i < count1; i++)
    {
        mergeList.Add(ob1[i]);
    }

    int count2 = ob2.getWorkersCount();
    for (size_t i = 0; i < count2; i++)
    {
        mergeList.Add(ob2[i]);
    }

    return mergeList;
}


int main()
{


    Worker** workers = new Worker * [3];
    workers[0] = new Worker("C", 0, 120, 8);
    workers[1] = new Worker("A", 0, 90, 12);
    workers[2] = new Worker("B", 2, 110, 6);

    WorkersList* lists = new WorkersList[3];
    lists[0] = WorkersList(workers, 3); //list constructor
```

```
    lists[0].Add(new Worker("E", 1, 80, 6)); //list add one

    Worker** workers2 = new Worker * [2];
    workers2[0] = new Worker("D", 1, 100, 170);
    workers2[1] = new Worker("G", 1);

    lists[1] = WorkersList();
    lists[1].Add(workers2, 2); // list add range

    lists[2] = lists[0] + lists[1]; // sum lists operator

      Worker* worker5 = lists[2].At(4);// list at (as index but not
throw out of range)

    worker5->Print();

    cout << "Print salaries:" << endl;
    float* slrs = lists[2].GetSalaries();
    int cnt = lists[2].getWorkersCount();
    for (size_t i = 0; i < cnt; i++)
    {
        cout << lists[2][i]->getName() << " "; //list index operator
        cout << slrs[i] << endl;
    }

    cout << "Print sorted list:" << endl;
    Worker** sortWorkers = lists[2].GetSortedWorkers();

    for (size_t i = 0; i < cnt; i++)
    {
        sortWorkers[i]->Print();
    }
```

```
    cout << "Print unsorted list:" << endl;
    cnt = lists[2].getWorkersCount();
    for (size_t i = 0; i < cnt; i++)
    {
        lists[2][i]->Print();
    }

    //delete worker
    lists[2].Delete(2);

    cout << "Print without deleted worker:" << endl;
    cnt = lists[2].getWorkersCount();
    for (size_t i = 0; i < cnt; i++)
    {
        lists[2][i]->Print();
    }

    return 0;
}
```

Тестирование программы

Тестирование задачи 1 представлено на рисунках 1, 2.

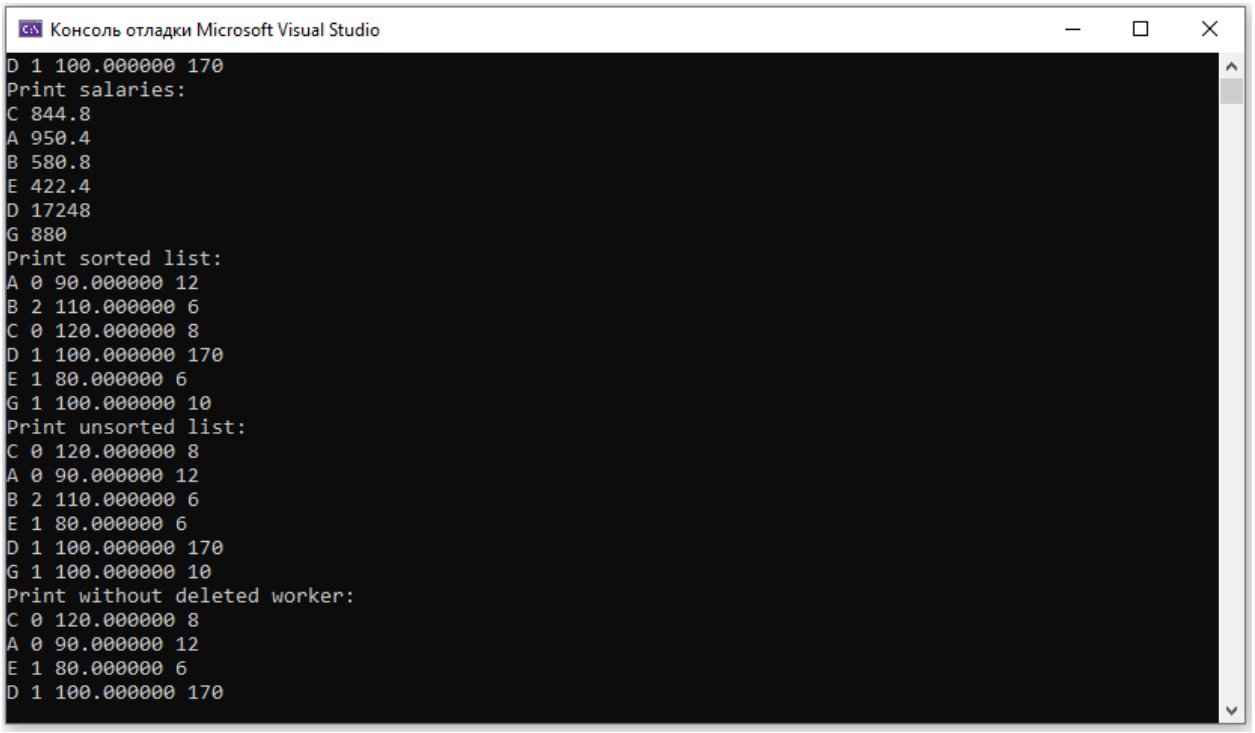

Рисунок 1 - Тест 1 задачи 1

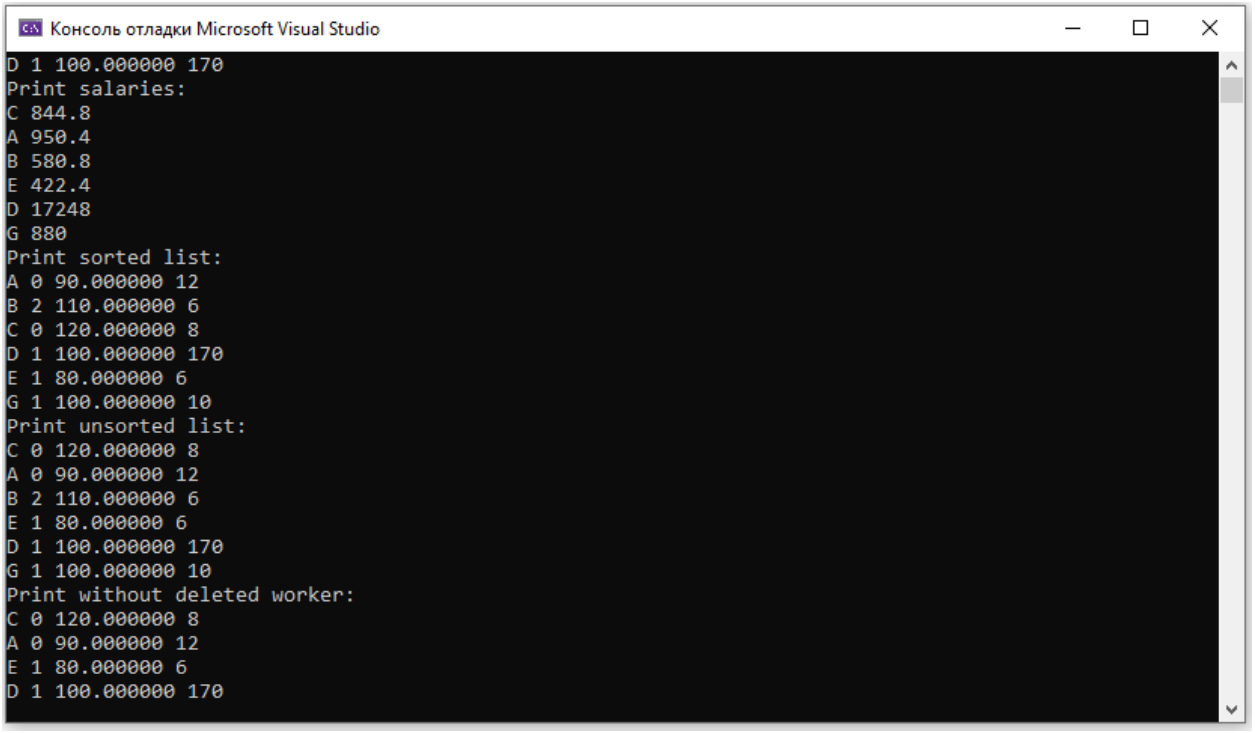

Рисунок 2 - Тест 2 задачи 1