МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р. Е. АЛЕКСЕЕВА»

Институт радиоэлектроники и информационных технологий
Кафедра «Прикладная математика»

Лабораторная работа №2

Выполнили:
Студенты группы 20-ПМ-1
Бугрова А.
Кирина Е.
Каретников Н.
Ермолаев Н.
Кочаровский Д.

Проверил:
Доцент кафедры «Прикладная математика»
Чернов А.Г.

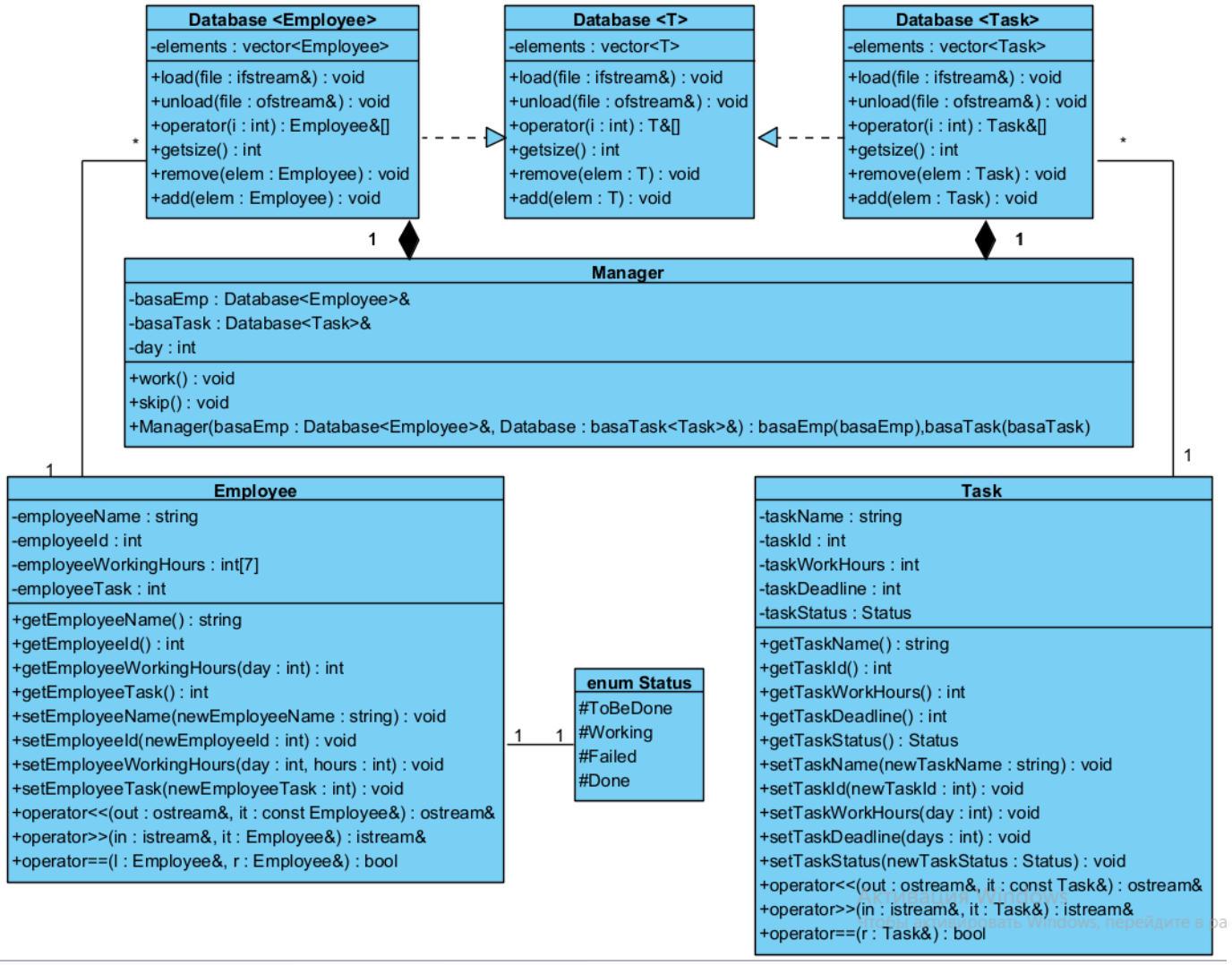
Нижний Новгород

2021

Содержание

Постановка задачи

UML-диаграмма



Описание классов в формате отчёта Doxygen

Task Manager

Титульная страница | Описания | Классы | Файлы

Класс Database

Класс работы с базой данных. Подробнее...

```
#include <Database.hpp>
```

Открытые члены

void	**load** (ifstream &file) Данная функция осуществляет загрузку базы данных из файла.
void	**unload** (ofstream &file) Данная функция осуществляет выгрузку базы данных из файла.
T	**operator[]** (int i)
int	**getsize** () Данная функция осуществляет взятия размеров массива.
void	**remove** (T elem) Данная функция осуществляет удаление элемента массива.

Подробное описание

Класс работы с базой данных.

Данный класс осуществляет функции для работы с базой данных.

Объявления и описания членов класса находятся в файле:

- D:/Projects/Project/**Database.hpp**

Класс Employee

Класс сотрудников Подробнее...

`#include <Employee.hpp>`

Открытые члены

string	**getEmployeeName** () Функция вывода имени сотрудника.
int	**getEmployeeId** () Функция вывода id сотрудника.
int	**getEmployeeWorkingHours** (int day) Функция вывода количества рабочих часов в неделю.
int	**getEmployeeTask** () Функция вывода задачи сотрудника.
void	**setEmployeeName** (string newEmployeeName) Функция присваивания имени новому сотруднику.
void	**setEmployeeId** (int newEmployeeId) Функция присваивания id новому сотруднику.
void	**setEmployeeWorkingHours** (int day, int hours) Функция присваивания количества рабочих часов в неделю новому сотруднику.
void	**setEmployeeTask** (int newEmployeeTask) Функция присваивания задачи новому сотруднику.

Друзья

ostream &	**operator<<** (ostream &out, const **Employee** &it)
istream &	**operator>>** (istream &in, **Employee** &it)
bool	**operator==** (**Employee** &l, **Employee** &r)

Подробное описание

Класс сотрудников

Данный класс отображает наличие у сотридника:имени,id,количества рабочих часов в неделю и его задачи.
Помимо этого,класс позволяет присвоить новое значение имени сотрудника,id сотрудника,количества рабочих часов в неделю и его задачу.

Объявления и описания членов классов находятся в файлах:

Класс Manager

Класс присваивания Подробнее...

```
#include <Manager.hpp>
```

Открытые члены

	Manager (**Database**< **Employee** > basaEmp, **Database**< **Task** > basaTask)
void	**work** ()
void	**skip** ()

Подробное описание

Класс присваивания

Данный класс обьявляет объекты для классов **Employee** и **Task**.
Также этот класс объявляет функции work и skip.

Объявления и описания членов классов находятся в файлах:

- D:/Projects/Project/**Manager.hpp**
- D:/Projects/Project/Manager.cpp

Класс Task

Класс задач для сотрудников. Подробнее...

```
#include <Task.hpp>
```

Открытые члены

string	**getTaskName** () Данная функция отвечает за вывод имени у задачи.
int	**getTaskId** () Данная функция отвечает за вывод id у задачи.
int	**getTaskWorkHours** (int day) Данная функция отвечает за вывод количества рабочих часов у задачи.
int	**getDeadline** () Данная функция отвечает за вывод оставшегося времени до сдачи задания.
Status	**getTaskStatus** () Данная функция отвечает за вывод статуса выполнения задачи сотрудником.
void setTask	**Name** (string newTaskName) Данная функция отвечает за присваивания имени новой задаче.
void	**setTaskId** (int newTaskId) Данная функция отвечает за присваивание id новой задаче.
void	**setTaskWorkHours** (int day) Данная функция отвечает за присваивание количества рабочих часов для новой задачи.
void	**setTaskDadline** (int days) Данная функция отвечает за присваивание количества оставшихся дней до сдачи задания.
void	**setTaskStatus** (int newTaskStatus) Данная функция отвечает за присваивание статуса выполнения для новой задачи.
bool	**operator==** (**Task** &r)

Друзья

ostream &	**operator<<** (ostream &out, const **Task** &it)
istream &	**operator>>** (istream &in, **Task** &it)

Подробное описание

Класс задач для сотрудников.

Класс **Task** отвечает за объявление функций для манипуляций задачами сотрудников.

Скриншоты работы программы

```
-
   company task planner

                                day 0
        Main menu
1) view database
2) find employee
3) find task
4) add manager entity
5) edit manager entity
6) delete manager entity
7) assign task to employee
8) pass some days

0)exit
1
choose the datadase
1- Employees
2- Tasks
1
Berezkin Dmitriy Alexandrovich
2 3 5 7 8 2 2 1 -1
Rizhov Dmitriy Alexandrovich
3 8 8 8 8 8 0 0 2
```

```
1) view database
2) find employee
3) find task
4) add manager entity
5) edit manager entity
6) delete manager entity
7) assign task to employee
8) pass some days

0)exit
2
enter employee's id
3
Rizhov Dmitriy Alexandrovich
3 8 8 8 8 8 0 0 2

                                    day 0
        Main menu
1) view database
2) find employee
3) find task
4) add manager entity
5) edit manager entity
6) delete manager entity
7) assign task to employee
8) pass some days

0)exit
```

Код программы

Main.cpp

```
#include <iostream>
#include <vector>
#include <fstream>
#include <string>
//#include "Database.hpp"
#include "Manager.hpp"
using namespace std;
/*void opening(ifstream filename, int n)//функция для проверки на открытие файлов с бд
{
	//file.open(filename);
	if (!filename.is_open())//если не открылся файл то выхоим с программы
	{
		cout << "   !!!   THE DATABASE NUMBER " << n << " IS BROKEN   !!!   " << endl;
		exit(0);
	}
}*/

int main(int argc, char** argv)
{
  if (argc!=3) //проверка на количество аргументов
  {
    cout << "   !!!   PLEASE, PROVIDE EXACTLY 2 DATABASE FILENAMES          !!!   " << endl;
    return 0;
  }
```

```
string employeeName=argv[1]; // переносим название бд в переменные
string taskName=argv[2];
ifstream file1(employeeName);
if (!file1.is_open())
{
   cout << "   !!!   THE DATABASE NUMBER 1 IS BROKEN   !!!   " << endl;
   exit(0);
}
ifstream file2(taskName);
if (!file2.is_open())
{
   cout << "   !!!   THE DATABASE NUMBER 2 IS BROKEN   !!!   " << endl;
   exit(0);
}



//opening(file1,1);// открываем файлы с бд
//opening(file2,2);


Database<Employee> basa1;// загружаем бд в оперативную память и
закрываем тектовые файлы
Database<Task> basa2;
basa1.load(file1);
basa2.load(file2);
file1.close();
file2.close();


Manager manager(basa1, basa2);
```

```
    manager.work();// запускаем работу менеджера


    ofstream file11(employeeName); //обновляем текстовые файлы бд и
закрываем их
    ofstream file22(taskName);
    if (!file11.is_open())
    {
        cout << "   !!!   THE DATABASE NUMBER 1 IS BROKEN   !!!   " << endl;
        exit(-1);
    }
    if (!file22.is_open())
    {
        cout << "   !!!   THE DATABASE NUMBER 2 IS BROKEN   !!!   " << endl;
        exit(-2);
    }
    //file11<< "dfghj \n";
    basa1.unload(file11);
    file11.close();


    basa2.unload(file22);
    file22.close();


    return 0;
}
```

Manager.hpp

```
#include <iostream>
#include "Database.hpp"
```

```
using namespace std;
class Manager
{
private:
   Database <Employee> &basaEmp;
   Database <Task> &basaTask;
   int day;
public:
   Manager(Database <Employee> &basaEmp, Database <Task>
&basaTask):basaEmp(basaEmp),basaTask(basaTask) {}
   void work();
   void skip();

};
```

Manager.cpp

```
#include "Manager.hpp"

using namespace std;

void Manager::skip()
{
   for (int i=0; i<basaEmp.getsize(); i++)
   {
      if (basaEmp[i].getEmployeeTask() != -1)
      {
         int code=basaEmp[i].getEmployeeTask();
         for (int j=0 ; j < basaTask.getsize() ; j++)
```

```
        {
            if (basaTask[j].getTaskId()==code)
            {

basaTask[j].setTaskWorkHours(basaTask[j].getTaskWorkHours()-basaEmp[i].getEmplo
yeeWorkingHours(day%7));
                break;
            }
        }
    }
  }
  for (int i=0; i<basaTask.getsize(); i++)
  {
    if (basaTask[i].getTaskWorkHours()<1)
    {
        basaTask[i].setTaskStatus(Status::Done);
        for (int i=0; i<basaEmp.getsize(); i++)
        {
            if (basaEmp[i].getEmployeeTask() == basaTask[i].getTaskId())
            {
                basaEmp[i].setEmployeeTask(-1);
            }
        }
    }
    if (basaTask[i].getTaskDeadline() == 0 && basaTask[i].getTaskWorkHours()>0 &&
basaTask[i].getTaskStatus() !=Status::Failed )
    {
        basaTask[i].setTaskStatus(Status::Failed);
    }
```

```
      if (basaTask[i].getTaskStatus()==Status::ToBeDone ||
basaTask[i].getTaskStatus()==Status::Working){
          basaTask[i].setTaskDeadline(basaTask[i].getTaskDeadline()-1);
      }
   }
   day++;
}

void Manager::work()
{
   cout << "   company task planner" << endl;
   char num='-1';
   int day=0;
   while (num!=0)
   {
      cout << endl;
      cout << "                              day "<<day;
      cout << "               \n";
      cout << "Main menu\n";
      cout << "1) view database\n";
      cout << "2) find employee\n";
      cout << "3) find task\n";
      cout << "4) add manager entity\n";
      cout << "5) edit manager entity\n";
      cout << "6) delete manager entity\n";
      cout << "7) assign task to employee\n";
      cout << "8) pass some days\n";
      cout << endl;
      cout << "0)exit\n";
```

```
cin >> num;
switch(num)
{
case '1':
{
   cout << "choose the datadase\n";
   cout << "1- Employees \n2- Tasks \n";
   int a;
   cin >> a;
   switch(a)
   {
   case 1:
   {
      for (int i=0; i< basaEmp.getsize(); i++)
      {
         cout << basaEmp[i] << endl;
      }
      break;
   }
   case 2:
   {
      for (int i=0; i< basaTask.getsize(); i++)
      {
         cout << basaTask[i] << endl;
      }
      break;
   }
   default:
```

```
    {
        cout << "try again :) \n";
        break;
    }
    }
    break;
}
case '2':
{
    cout << "enter employee's id \n";
    int code;
    cin >> code;
    for (int i=0; i<basaEmp.getsize(); i++)
    {
        if (basaEmp[i].getEmployeeId()==code)
        {
            cout << basaEmp[i] << endl;
            break;
        }
    }
    break;
}
case '3':
{
    cout << "enter task's id \n";
    int code;
    cin >> code;
    for (int i=0; i<basaTask.getsize(); i++)
    {
```

```
        if (basaTask[i].getTaskId()==code)
        {
            cout << basaTask[i] << endl;
            break;
        }
    }
    break;
}
case '4':
{
    cout << "choose the entity\n";
    cout << "1- Employee \n2- Task \n";
    int a;
    cin >> a;
    switch(a)
    {
    case 1:
    {
        Employee temp;
        cout << "enter the name of employee \n";
        string name;
        getline(cin,name);
        getline(cin,name);
        temp.setEmployeeName(name);
        int id(basaEmp.getsize()+1);
        temp.setEmployeeId(id);
        cout << "enter work hours \n";
        int hour;
        for (int i=0; i<7; i++)
```

```
    {
        cin >> hour;
        temp.setEmployeeWorkingHours(i,hour);
    }
    temp.setEmployeeTask(-1);
    basaEmp.add(temp);
    break;
}
case 2:
{
    Task temp;
    cout << "enter the name of task \n";
    string name;
    getline(cin,name);
    getline(cin,name);
    temp.setTaskName(name);
    int id(basaTask.getsize()+1);
    temp.setTaskId(id);
    cout << "enter hours for working \n";
    int hour;
    cin >> hour;
    temp.setTaskWorkHours(hour);
    cout << "enter days to deadline \n";
    int days;
    cin >> days;
    temp.setTaskDeadline(days);
    temp.setTaskStatus(Status::ToBeDone);
    basaTask.add(temp);
    break;
```

```
    }
    default:
    {
        cout << "try again :) \n";
        break;
    }
    }
    break;
}
case '5':
{
    cout << "choose the entity you want to edit\n";
    cout << "1- Employee \n2- Task \n";
    int a;
    cin >> a;
    switch(a)
    {
    case 1:
    {
        cout << "employee's id \n";
        int code;
        cin >> code;
        Employee temp;
        for (int i=0; i<basaEmp.getsize(); i++)
        {
            if (basaEmp[i].getEmployeeId()==code)
            {
                temp=basaEmp[i];
            }
```

```
    }
    cout << "enter new work hours \n";
    int hour;
    for (int i=0; i<7; i++)
    {
        cin >> hour;
        temp.setEmployeeWorkingHours(i,hour);
    }
    break;
}
case 2:
{
    cout << "task's id \n";
    int code;
    cin >> code;
    Task temp;
    for (int i=0; i<basaTask.getsize(); i++)
    {
        if (basaTask[i].getTaskId()==code)
        {
            temp=basaTask[i];
        }
    }
    cout << "enter days to deadline \n";
    int days;
    cin >> days;
    temp.setTaskDeadline(days);
    break;
}
```

```
    default:
    {
        cout << "try again :) \n";
        break;
    }
    }
    break;
}
case '6':
{
    cout << "choose the entity you want to delete\n";
    cout << "1- Employee \n2- Task \n";
    int a;
    cin >> a;
    switch(a)
    {
    case 1:
    {
        cout << "enter employee's id \n";
        int code;
        cin >> code;
        for (int i=0; i<basaEmp.getsize(); i++)
        {
            if (basaEmp[i].getEmployeeId()==code)
            {
                basaEmp.remove(basaEmp[i]);
            }
        }
        break;
```

```
    }
    case 2:
    {
        cout << "enter task's id \n";
        int code;
        cin >> code;
        for (int i=0; i<basaTask.getsize(); i++)
        {
            if (basaTask[i].getTaskId()==code)
            {
                basaTask.remove(basaTask[i]);
            }
        }
        break;
    }
    default:
    {
        cout << "try again :) \n";
        break;
    }
    }
    break;
}
case '7':
{
    cout << "enter employee's id and task's id \n";
    int ecode,tcode;
    cin >> ecode >> tcode;
    for (int i=0; i<basaEmp.getsize(); i++)
```

```
    {
        cout << "0000";
        if (basaEmp[i].getEmployeeId()==ecode)
        {
            cout << 111;
            basaEmp[i].setEmployeeTask(tcode);
            basaTask[i].setTaskStatus(Status::Working);
            break;
        }
    }
    break;
}
case '8':
{
    cout << "enter number of days \n";
    int d;
    cin >> d;
    for (int i=0; i<d; i++)
    {
        skip();
    }
    day+=d;
    break;
}
case '0':
{
    cout << "have a good day \n";
    return;
}
```

```
        default:
        {
            cout << "please choose correct command \n";
        }
        }
    }
}
```

Database.hpp

```
#include <vector>
#include "Employee.hpp"
#include "Task.hpp"

using namespace std;

template <typename T>

class Database
{
private:
    vector <T> elements;
public:
    void load (ifstream& file)
    {
        T temp;
        //cout << "fnfuif\n";
        while (file>>temp)
```

```
    {
        //cout << temp << endl;
        elements.push_back(temp);
    }
}
void unload(ofstream& file)
{
    //cout << 1;
    for (int i=0; i<elements.size(); i++)
    {
        //cout << elements[i] << endl;
        file << elements[i] << endl;
    }
}
T& operator[] (int i)
{
    return elements.at(i);
}
int getsize()
{
    return elements.size();
}
void remove(T elem)
{
    for (int i=0; i< elements.size(); i++)
    {
        if (elements[i]==elem)
        {
            elements.erase(elements.begin()+i);
```

```
            break;
          }
        }
    }
  void add(T elem){
        elements.push_back(elem);
    }
};
```

Employee.hpp

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

using namespace std;

class Employee
{
private:
    string employeeName;
    int employeeId;
    int employeeWorkingHours[7];
    int employeeTask;
public:
    string getEmployeeName();
    int getEmployeeId();
    int getEmployeeWorkingHours(int day);
    int getEmployeeTask();
    void setEmployeeName(string newEmployeeName);
    void setEmployeeId(int newEmployeeId);
```

```
    void setEmployeeWorkingHours(int day, int hours);
    void setEmployeeTask(int newEmployeeTask);
    friend ostream& operator<< (ostream &out, const Employee &it);
    friend istream& operator>> (istream &in, Employee &it);
    friend bool operator== (Employee& l, Employee& r);
};
```

Employee.cpp

```
#include "Employee.hpp"
using namespace std;

string Employee::getEmployeeName()
{
    return employeeName;
}
int Employee::getEmployeeId()
{
    return employeeId;
}
int Employee::getEmployeeWorkingHours(int day)
{
    return employeeWorkingHours[day];
}
int Employee::getEmployeeTask()
{
    return employeeTask;
}
void Employee::setEmployeeName(string newEmployeeName)
{
```

```
    this->employeeName = newEmployeeName;
}
void Employee::setEmployeeId(int newEmployeeId)
{
    this->employeeId = newEmployeeId;
}
void Employee::setEmployeeWorkingHours(int day, int hours)
{
    this->employeeWorkingHours[day] = hours;
}
void Employee::setEmployeeTask(int newEmployeeTask)
{
    this->employeeTask = newEmployeeTask;
}
ostream& operator<< (ostream& out, const Employee& it)
{
    out << it.employeeName << endl << it.employeeId << " ";
    for (int i = 0; i < 7; i++)
    {
        out << it.employeeWorkingHours[i] << " ";
    }
    out << it.employeeTask;
    return out;
}
istream& operator>> (istream& in, Employee& it)
{
    getline(in,it.employeeName);
    in >> it.employeeId;
    for (int i = 0; i < 7; i++)
```

```
    {
        in >> it.employeeWorkingHours[i];
    }
    in >> it.employeeTask;
    string trash;
    getline(in,trash);
    return in;
}
bool operator== (Employee& l, Employee& r)
{
    bool check = true;
    for (int i = 0; i < 7; i++)
    {
        if (l.employeeWorkingHours[i] != r.employeeWorkingHours[i])
        {
            check = false;
            break;
        }
    }
    return l.employeeId == r.employeeId &&
           l.employeeName == r.employeeName &&
           l.employeeTask == r.employeeTask &&
           check;

}
```

Task.hpp

```
#include <fstream>
#include <string>
```

```
#include <iostream>
using namespace std;
enum class Status
{
   ToBeDone,
   Working,
   Failed,
   Done
};
class Task
{
private:
   string taskName;
   int taskId;
   int taskWorkHours;
   int taskDeadline;
   Status taskStatus;
public:
   string getTaskName();
   int getTaskId();
   int getTaskWorkHours();
   int getTaskDeadline();
   Status getTaskStatus();
   void setTaskName(string newTaskName);
   void setTaskId(int newTaskId);
   void setTaskWorkHours(int day);
   void setTaskDeadline(int days);
   void setTaskStatus(Status newTaskStatus);
   friend ostream& operator<< (ostream &out, const Task &it);
```

```
    friend istream& operator>> (istream &in, Task &it);
    bool operator== (Task& r);
};
```

Task.cpp

```
#include "Task.hpp"
#include <sstream>
using namespace std;

string Task::getTaskName()
{
    return this->taskName;
}
int Task::getTaskId()
{
    return this->taskId;
}
int Task::getTaskWorkHours()
{
    return this->taskWorkHours;
}
int Task::getTaskDeadline()
{
    return this->taskDeadline;
}
Status Task::getTaskStatus()
{
```

```
    return this->taskStatus;
}
void Task::setTaskName(string newTaskName)
{
    this->taskName = newTaskName;
}
void Task::setTaskId(int newTaskId)
{
    this->taskId = newTaskId;
}
void Task::setTaskWorkHours(int day)
{
    this->taskWorkHours = day;
}
void Task::setTaskDeadline(int day)
{
    this->taskDeadline = day;
}
void Task::setTaskStatus(Status newTaskStatus)
{
    this->taskStatus = newTaskStatus;
}


ostream& operator<< (ostream& out, const Task& it)
{
    out<< it.taskName <<'\n'<< it.taskId << ' ' << it.taskWorkHours << ' ' <<
it.taskDeadline << '\n';
    if (it.taskStatus==Status::ToBeDone)
    {
```

```
        out<<"ToBeDone";
    }
    if (it.taskStatus==Status::Failed)
    {
        out<< "Failed";
    }
    if (it.taskStatus==Status::Done)
    {
        out<<"Done";
    }
    if (it.taskStatus==Status::Working)
    {
        out<<"Working";
    }
    return out;
}

istream& operator>> (istream& in, Task& it)
{
    getline(in,it.taskName);
    string s;
    getline(in,s);
    stringstream ss(s);
    ss>>it.taskId >> it.taskWorkHours >> it.taskDeadline;
    getline(in,s);
    if (s=="ToBeDone")
    {
        it.taskStatus=Status::ToBeDone;
    }
```

```
    if (s=="Failed")
    {
        it.taskStatus=Status::Failed;
    }
    if (s=="Done")
    {
        it.taskStatus=Status::Done;
    }
    if (s=="Working")
    {
        it.taskStatus=Status::Working;
    }
    return in;
}

bool Task::operator==(Task& r)
{
    if (this->taskName == r.taskName && this->taskId == r.taskId &&
this->taskWorkHours == r.taskWorkHours && this->taskDeadline == r.taskDeadline
&& this->taskStatus == r.taskStatus) return true;
    return false;
}
```

Makefile

```
CFLAGS=-c -Wall

all: taskManager

taskManager: main.o Manager.o Employee.o Task.o
        g++ main.o Manager.o Employee.o Task.o -o taskManager
```

```
main.o: main.cpp
	g++ $(CFLAGS) main.cpp

Manager.o: Manager.cpp
	g++ $(CFLAGS) Manager.cpp

Employee.o: Employee.cpp
	g++ $(CFLAGS) Employee.cpp

Task.o: Task.cpp
	g++ $(CFLAGS) Task.cpp

clean:
	rm -rf *.o taskManager
```