Advanced Programming

Homework Assignment 4

Operators, friend, Files

<u>General guidelines:</u>

a. Maximize readability and ensure indentation.
b. Do exactly as required in the questions.
c. Every class in every question must be submitted in two seperate files – a header file (.h) and an implementation file (.cpp).
d. Add functions and helper methods as necessary to ensure readability.
e. Submission must be done according to the <u>submission guidelines</u> – a document published in the course website – read them carefully!
f. Use relevant names.
g. Use comments to document your functions and complex code parts. **Add an output example at the end of your file!**
h. Individual work and submission – paired submission is not allowed.

Important note: Unless otherwise specified, ever homework has <u>no more than one week</u> for submission.
<u>Open submission boxes past the deadline are not permission for late submission.</u>

<u>Question 1:</u>

<u>Overview</u>

This question has the following objectives:

1. Implementation of a class called **Worker** that represents a worker (id, name, wages).
2. Implementation of a class called **WorkersFile** that operates a file containing worker data, with the following capabilities:
   a. **Input** of a Worker into the file.
   b. **Printing** of all the Workers in the file.
   c. **Updating** all the Workers in the file to add a **bonus** to their wages.
   d. **Sorting** all the Workers in the file according to wages (in increasing/decreasing order).
   e. Extraction of a Worker according to the **location in the file**.
   f. **Merging** two WorkerFile files into a new WorkerFile file.

<u>Note:</u> In this question you are required to implement both class methods and friend functions. The **friend functions** are to be implemented in the **same** separate cpp file that the **class methods** are implemented.

Tip: Before solving the question it might help to read the execution examples and the given main program to better understand the requirements.

<u>Part A – The Worker Class:</u>

Define and implement a class called **Worker** that represents a worker

The class must include the following fields (**private**):

- A positive **id number** (long) – greater then 0, doesn't begin with 0.
- The worker's **name** (string) – no spaces.

- The monthly **wages** (float) – greater than 0.

Note: Even though the fields are private **do not implement getters and setters**. Assume the values are valid (**no validity check is required**).

The fields must be set to the following default values:

- The id number and the wages should be 0 by default (an unassigned worker).
- The name will be an empty string (“”).

You may assign the default values in the field declaration or in a no-parameter constructor. **You may not implement any other constructors.**

The class must include only the following **public methods**:

- **operator+=**
  - Receives a float argument that represents a bonus.
  - Updates the Worker instance so that the bonus is added to the wages.
  - Returns a reference to the updated worker.
- **operator<**
  - Receives another Worker as a right operand (const and &).
  - Returns true if the left operand wages are less than the right operand wages. Otherwise returns false.

The following **public friend methods** must be defined for the class:

- **operator==** equality to left operand of type **long**
  - Receives a Worker as a right operand (const and &).
  - Returns true if the id numbers of both operands are equal. Otherwise returns false.
- **operator==** equality to left operand of type **string**
  - Receives a Worker as a right operand (const and &).
  - Returns true if the names of both operands are equal. Otherwise returns false.
- **operator>>** input from **istream**&
  - Receives a Worker as a right operand (const and &).
  - Inputs the values of the three fields (consecutively, separated by spaces) into the Worker instance.
  - Returns a reference to the input object (istream&).
- **operator<<** output to **ostream**&
  - Receives a Worker as a right operand (const and &).
  - Outputs the values of the three fields (consecutively, separated by spaces) from the Worker instance.
  - Returns a reference to the output object (ostream&).

Note: In the main and other external functions i/o of workers (e.g. using the console or files) will be done using **only** the >> and << operators you implemented.

Part B – The WorkersFile class:

The class described below makes use of an enum that describes the state of a file. Copy to the class header file before the class declaration the following enum:

```
enum FILE_STATUS { ERROR, CLOSED, OPEN_R, OPEN_W };
```

Define and implement a class called **WorkersFile** that operates a physical file that contains worker data. The class **WorkersFile** will make use of **Worker** instances and use as necessary the methods you implemented for that class.

Example of the content of a file test.txt that contains the data (id number, name, wages) of 3 workers:

```
1111 dan 6500
11 gad 2200.55
111 shimon 3000
```

Every worker is on a separate line with spaces between the field values.

The class must include the following fields (**private**):

- **fileName** (string) – the name of the file.
- **iofile** (fstream) – the object that will handle the file.
- **status** (FILE_STATUS) – the state of the file: error, closed, open for reading or open for writing.
- **size** (int) – the current number of workers stored in the file.

Note:

- The file is to be opened as necessary before each operation and closed at the end of each operation. The field **status** should be modified accordingly.
- After the input of each worker the field **size** should be modified.
- Even though the fields are private **do not implement getters and setters**. Assume the values are valid (**no validity check is required**).

The class must include the following **helper** methods (**private**):

- **openNewFileForWriting** – a **boolean** helper method that opens the file for **writing** (using the field **iofile**) and updates the field **status** accordingly:
  - If the status was not CLOSED print "ERROR" and return false.
  - If the file opened successfully status will become OPEN_W and return true.
  - If the file failed to open successfully status will become ERROR, print "ERROR" and return false.
- **openFileForReading** – a **boolean** helper method that opens an existing file for **reading** (using the field **iofile**) and updates the field **status** accordingly:
  - If the status was not CLOSED print "ERROR" and return false.
  - If the file opened successfully status will become OPEN_R and return true.
  - If the file failed to open successfully status will become ERROR, print "ERROR" and return false.
- **closeFile** – a helper method that closes an open file (using the field **iofile**) and sets the field **status** to CLOSED.
- You may add more private helper methods if necessary.

The class must include the following methods (**public**):

- **Assignment constructor** – receives an instance of string that contains a filename and initializes the fields thus:
  - **fileName** will be assigned the value of the argument.
  - **size** will be assigned 0.
  - **status** will be set to CLOSED.
  - A new empty file is to be created and closed with the name provided in the argument. This can be done by calling openNewFileForWriting and then calling closeFile.

- **Copy constructor** – receives a **WorkersFile** instance and copies the fields thus:
  - **fileName** and **size** will be copies as is.
  - **status** will be set to CLOSED.
  - **No other change is necessary.** It is not required to create a file because a file was created for the source instance.
- **Destructor** – calls closeFile if for some reason the file is open.

The following **public friend methods** must be defined for the class:

- **operator>>** input from **istream&**
  - Receives a WorkersFile as a right operand (& but not const).
  - Inputs a loop of Workers where the values of the three fields are inputted consecutively separated by spaces, and writes each of the the workers into the physical file represented by the field **iofile** belonging to the right operand.
  - The input loop terminates after an **empty worker** (with id number 0) is inputted. The following == test can be used (with the operator defined in Worker):

    ```
    Worker w; //...
    if (0 == w) //...
    ```
  - Each worker will be inputted using the input operator>> defined in Worker.
  - Returns a reference to the input object (istream&).
- **operator<<** output to **ostream&**
  - Receives a WorkersFile as a right operand (& but not const).
    - Note: Usually the right operand of this operator is const. Because in this particular case the fstream needs to be modified throughout it cannot be const here.
  - Reads all the workers in the physical file represented by the field **iofile** and outputs them. Each worker is to be saved in a Worker instance and then outputted using the operator<< implemented for Worker.
  - Returns a reference to the output object (ostream&).

Note: In the main and other external functions i/o of workers (e.g. using the console or files) will be done using **only** the >> and << operators you implemented.

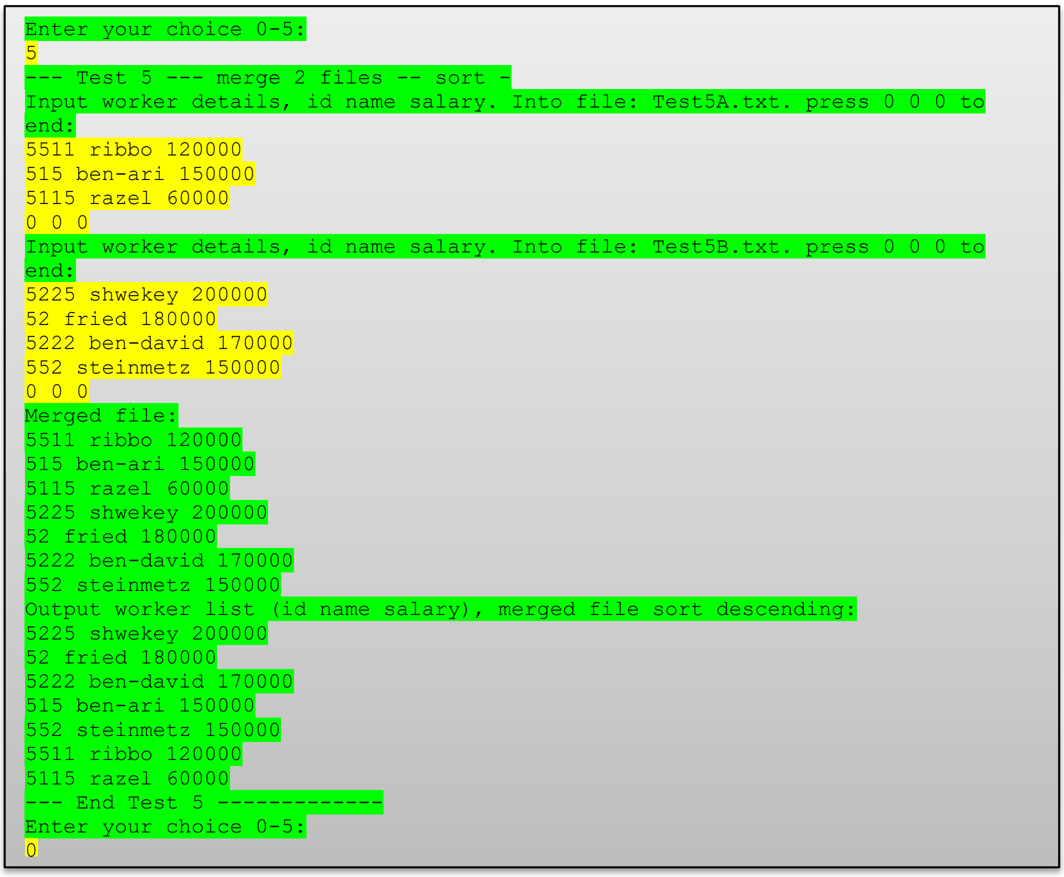
The class must include only the following **public methods**:

- **sort** – sorts the current file according to wages.
  - Receives a boolean parameter that defaults to true. When the argument is true the order is to be ascending and when it is false the order is to be descending.
  - The method reads all the workers into a local array of Workers, sorts them in the array and the writes them sorted back into the file.
- **operator[]** – extracts a Worker for a location in the current file.
  - Receives the index of a worker.
  - Returns a Worker instance with the fields set to the values in the corresponding line in the file.
    - Note: Usually operator[] returns a reference to allow for modification and to prevent copying needlessly. In this case because the access is to a physical file this is not possible and a Worker instance must be returned (Worker, not Worker&).
- **operator+=** – adds a bonus to the wages of the workers in the file.
  - Receives a right operand float as the bonus for all the workers.

  - The method reads all the workers into a local array of Workers, adds the bonus to each of the workers and writes them into the file with the new values.
  - The method returns a reference to the WorkersFile instance (in its state after the modification).
- **operator+** – merges with another file.
  - Receives a WorkersFile reference as the right operand.
  - The method creates and returns a new instance of WorkersFile which contains the workers from the right operand file appended to the workers from the left operand file (first left then right).
    - **Loops are not allowed in this method.** Use other operators that you implemented.
    - Note: Usually the + operator expects both operands to be const. Because both iofile fields must be modified for the reading this is not an option here and neither can be const.

Output example 1:

Input of two files, merging, sorting of the merged file



```
Enter your choice 0-5:
5
--- Test 5 --- merge 2 files -- sort -
Input worker details, id name salary. Into file: Test5A.txt. press 0 0 0 to
end:
5511 ribbo 120000
515 ben-ari 150000
5115 razel 60000
0 0 0
Input worker details, id name salary. Into file: Test5B.txt. press 0 0 0 to
end:
5225 shwekey 200000
52 fried 180000
5222 ben-david 170000
552 steinmetz 150000
0 0 0
Merged file:
5511 ribbo 120000
515 ben-ari 150000
5115 razel 60000
5225 shwekey 200000
52 fried 180000
5222 ben-david 170000
552 steinmetz 150000
Output worker list (id name salary), merged file sort descending:
5225 shwekey 200000
52 fried 180000
5222 ben-david 170000
515 ben-ari 150000
552 steinmetz 150000
5511 ribbo 120000
5115 razel 60000
--- End Test 5 -------------
Enter your choice 0-5:
0
```

Output example 2:

File input, sorting ascending and descending, printing the maximal and minimal wage workers

```
Enter your choice 0-5:
3
--- Test 3 --- sort -- operator[] --
Input worker details, id name salary. Into file: Test3.txt. press 0 0 0 to end:
33 sara 10000
3333 rivka 5000
3 rachel 7000
333 lea 4000
330 yael 2500
0 0 0
Output worker list (id name salary), before sorting. From file: Test3.txt
33 sara 10000
3333 rivka 5000
3 rachel 7000
333 lea 4000
330 yael 2500
Output worker list (id name salary), after sort ascending. From file: Test3.txt
330 yael 2500
333 lea 4000
3333 rivka 5000
3 rachel 7000
33 sara 10000
The worker with the minimum salary: 330 yael 2500
Output worker list (id name salary), after sort descending. From file:
Test3.txt
33 sara 10000
3 rachel 7000
3333 rivka 5000
333 lea 4000
330 yael 2500
The worker with the maximum salary: 33 sara 10000
--- End Test 3 -------------
Enter your choice 0-5:
0
```

Use the following main program to test your code:

```
#include "WorkersFile.h"
#include <iostream>
using namespace std;

//declarations of help functions
void Test1();
void Test2();
void Test3();
void Test4();
void Test5();

enum OPTIONS { STOP, TEST1, TEST2, TEST3, TEST4, TEST5 };

int main()
{
        int choice;
        do
        {
                cout << "Enter your choice 0-5:" << endl;
                cin >> choice;
                switch (choice)
```

```
			{
			case TEST1:
				Test1();
				break;
			case TEST2:
				Test2();
				break;
			case TEST3:
				Test3();
				break;
			case TEST4:
				Test4();
				break;
			case TEST5:
				Test5();
				break;
			default:
				break;
			}
	} while (choice);
}


void Test1()
{
	cout << "--- Test 1 --- one Worker operators --" << endl;

	Worker w1;
	cout << "Empty Worker: " << w1;

	cout << "Input first worker details, id name salary:" << endl;
	cin >> w1;

	cout << "First Worker: " << w1;

	w1 += 350.4;
	cout << "Worker after bonus: " << w1;

	Worker w2;
	cout << "Input second worker details, id name salary:" << endl;
	cin >> w2;

	cout << "Second Worker: " << w2;

	if (w1 < w2)
		cout << "First worker is smaller than second worker" <<
endl;

	if (w2 < w1)
		cout << "Second worker is smaller than second worker" <<
endl;

	int id;
	cout << "Input worker id: " << endl;
	cin >> id;

	if (id == w2)
		cout << "Id of second worker is " <<  id << endl;
	else
		cout << "Id of second worker is not " << id << endl;

	string name;
	cout << "Input worker name: " << endl;
	cin >> name;
	if (name == w1)
		cout << "Name of first worker is " << name << endl;
	else
		cout << "Name of first worker is not " << name << endl;

	cout << "--- End Test 1 -------------" << endl;
}

void Test2()
{
	cout << "--- Test 2 --- cout -- cin --" << endl;

	string name = "Test2.txt";
	WorkersFile wf2(name);

	cout << "Input worker details, id name salary. Into file: " <<
name << ". press 0 0 0 to end:" << endl;
	cin >> wf2;

	cout << "Output worker list (id name salary). From file: " << name
<< "" << endl;
	cout << wf2;

	cout << "Input worker details, id name salary. Into file: " <<
name << ". press 0 0 0 to end:" << endl;
	cin >> wf2;

	cout << "Output worker list (id name salary). From file: " << name
<< "" << endl;
	cout << wf2;

	cout << "--- End Test 2 -------------" << endl;
}


void Test3()
{
	cout << "--- Test 3 --- sort -- operator[] --" << endl;

	string name = "Test3.txt";
	WorkersFile wf3(name);

	cout << "Input worker details, id name salary. Into file: " <<
name << ". press 0 0 0 to end:" << endl;
	cin >> wf3;

	cout << "Output worker list (id name salary), before sorting. From
file: " << name << "" << endl;
	cout << wf3;

	wf3.sort(); //sort ascending
	cout << "Output worker list (id name salary), after sort
ascending. From file: " << name << "" << endl;
	cout << wf3;

	cout << "The worker with the minimum salary: " << wf3[0];

	wf3.sort(false); //sort descending
	cout << "Output worker list (id name salary), after sort
descending. From file: " << name << "" << endl;
	cout << wf3;

	cout << "The worker with the maximum salary: " << wf3[0];
	cout << "--- End Test 3 -------------" << endl;
}

void Test4()
{
	cout << "--- Test 4 --- operator+= --" << endl;

	string name = "Test4.txt";
	WorkersFile wf4(name);

	cout << "Input worker details, id name salary. Into file: " <<
name << ". press 0 0 0 to end:" << endl;
	cin >> wf4;

	cout << "Output worker list (id name salary), before bonus. From
file: " << name << "" << endl;
	cout << wf4;

	wf4 += 1000;
	cout << "Output worker list (id name salary), after bonus. From
file: " << name << "" << endl;
	cout << wf4;

	cout << "--- End Test 4 -------------" << endl;
}

void Test5()
{
	cout << "--- Test 5 --- merge 2 files -- sort -" << endl;

	string name5a = "Test5A.txt";
	WorkersFile wf5_a(name5a);

	cout << "Input worker details, id name salary. Into file: " <<
name5a << ". press 0 0 0 to end:" << endl;
	cin >> wf5_a;

	string name5b = "Test5B.txt";
	WorkersFile wf5_b(name5b);
	cout << "Input worker details, id name salary. Into file: " <<
name5b << ". press 0 0 0 to end:" << endl;
	cin >> wf5_b;

	WorkersFile wf5 = wf5_a + wf5_b; //the compiler optimized by not
calling to copy ctor
	cout << "Merged file:" << endl;
	cout << wf5;

	wf5.sort(false);
	cout << "Output worker list (id name salary), merged file sort
descending:" << endl;
	cout << wf5;

	cout << "--- End Test 5 -------------" << endl;
}
```

Good Luck!