(1) • Modify the implementation of the Heap class posted on CCLE so that it stores

`std::pair<string,int>`

not integers. Call the new class `HeapTasks`. In each `pair` the `string` will contain a task to be done and `int` the priority of the task. When you implement the class `HeapTask`, make sure the following code can be compiled and executed:

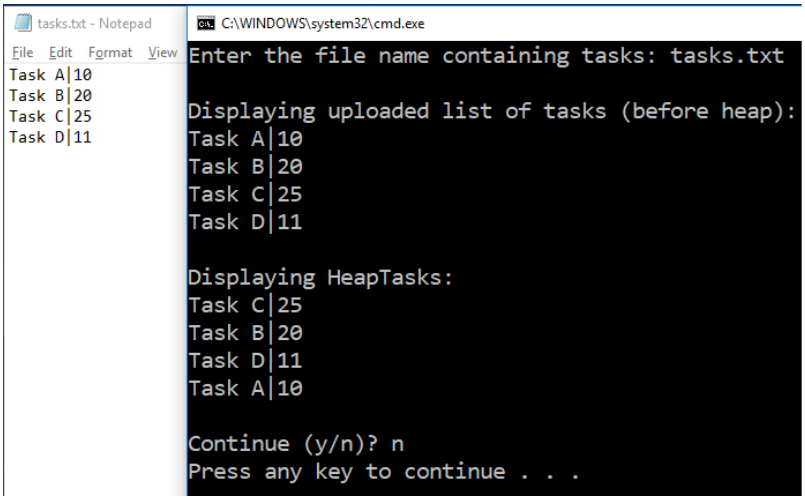
```
#include <iostream>
using namespace std;
...
HeapTasks tasks;
tasks.push(make_pair("Task A", 10));
tasks.push(make_pair("Task B", 100));
tasks.push(make_pair("Task C", 20));

while ( tasks.size() > 0 )
{
        pair<string,int> task = tasks.top();
        tasks.pop();
        cout << task.first << ", " << task.second << "\n";
}
```

• Write a program that reads tasks from a text file and then constructs `HeapTasks`. After `HeapTasks` is constructed its values should be displayed one by one using `top()` followed by `pop()` member function. Implement a loop in which the above actions are repeated until the user requests to quit. For the format of tasks see the input-output sample.

• Submit your solution as `hmw_7_1.cpp`.

• Sample input-output (see next page):



(2) Matrix class. Design a class `Matrix` which allows for addition of two matrices, multiplication of a matrix by a constant, multiplication of a matrix by a vector, and multiplication of two matrices using overloaded operators `*,+,-`. In addition, one must be able to print out an objest of the `Matrix` class using `std::cout`. Finally, the *getter* and *setter* of each entry in the matrix must be implemented using the overloaded member function `operator()`; for details see the explanation below.

Explanation.

1. A matrix is a rectangular array of numbers arranged in rows and columns. An $n \times m$ matrix A has n rows and m columns and is typically written in the form

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{bmatrix}$$

   The entry in the matrix A located in the i-th row and j-tj column is denoted by a_{ij}.

2. Given a real-valued constant $c \in \mathbb{R}$ and $n \times m$ matrix A, the matrix $B = A * c$ is obtained by multiplying each entry of A by c, that is, $b_{ij} = a_{ij} * c$.

3. Given two $n \times m$ matrices A and B, the sum $D = A + B$ is the new matrix with entries $d_{ij} = a_{ij} + b_{ij}$.

4. Given vector $x = (x_1, x_2, ..., x_m)$ of length m and $n \times m$ matrix A, the vector $y = A * x$ has length n and satisfies $y_i = \sum_{k=1}^{m} a_{ik} x_k$, where $i = 1, \dots, n$.

5. Given an $n \times m$ matrix A and $m \times k$ matrix B, the matrix $D = A * B$ has dimensions $n \times k$ and its entries are given by $d_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj}$.

Instructions and output.

- Design the `Matrix` class. A possible interface which you can start with is the following:

```
1  #include<iostream>
2  #include<vector>
3  #include<iomanip>
4  using namespace std;
5
6  class Matrix
7  {
8  public:
9          //nickname for the data type storing size of the vector
10         typedef vector<double>::size_type size_type;
11
12         //nickname for vector<double>
```

```
13          typedef vector<double> Vector;
14
15          // constructors
16          Matrix(){} // empty matrix
17          Matrix( size_type n, size_type m, double val = 0.0); // n x m matrix
18          Matrix( const Vector & v); // matrix n x 1 derived from Vector
19
20          //operator() for getting Matrix (i,j) value
21          double operator()(size_type i, size_type  j) const;
22
23          //operator() for setting Matrix (i,j) value
24          double & operator()(size_type i, size_type  j);
25
26          //operator*:
27          Matrix operator*(double c) const; // Matrix*constant
28          vector<double> operator*(const vector<double> &v) const; // Matrix*Vector
29          Matrix operator*(const Matrix & B) const; //Matrix*Matrix
30
31          //operator+:
32          Matrix operator+(const Matrix & B) const;//Matrix+Matrix
33          Matrix operator-(const Matrix & B) const;//Matrix-Matrix
34          Matrix operator-() const;// -Matrix
35
36          //matrix size getters (2 alternatives):
37          pair< size_type, size_type > size() const; // returns std::pair
38          void size( size_type & n, size_type & m ) const; //size is set in n, m
39
40          //printing the matrix as a table:
41          //Here width and prec are formatting parameters used for the input.
42          //width is used in cout << setw(bw)
43          //prec  is used in cout << fixed << setprecision(prec)
44          //width=6 and prec = 2 are default values
45          void print(unsigned short width = 6, unsigned short prec = 2) const;
46
47          //returns true if the matrix is empty
48          bool empty() const;
49
50          //resizing matrix to n x m
51          void resize(size_type n, size_type m );
52
53          //needed for overloading << operator which uses print()
54          friend ostream & operator<<( ostream& os, const Matrix & A );
55  private:
```

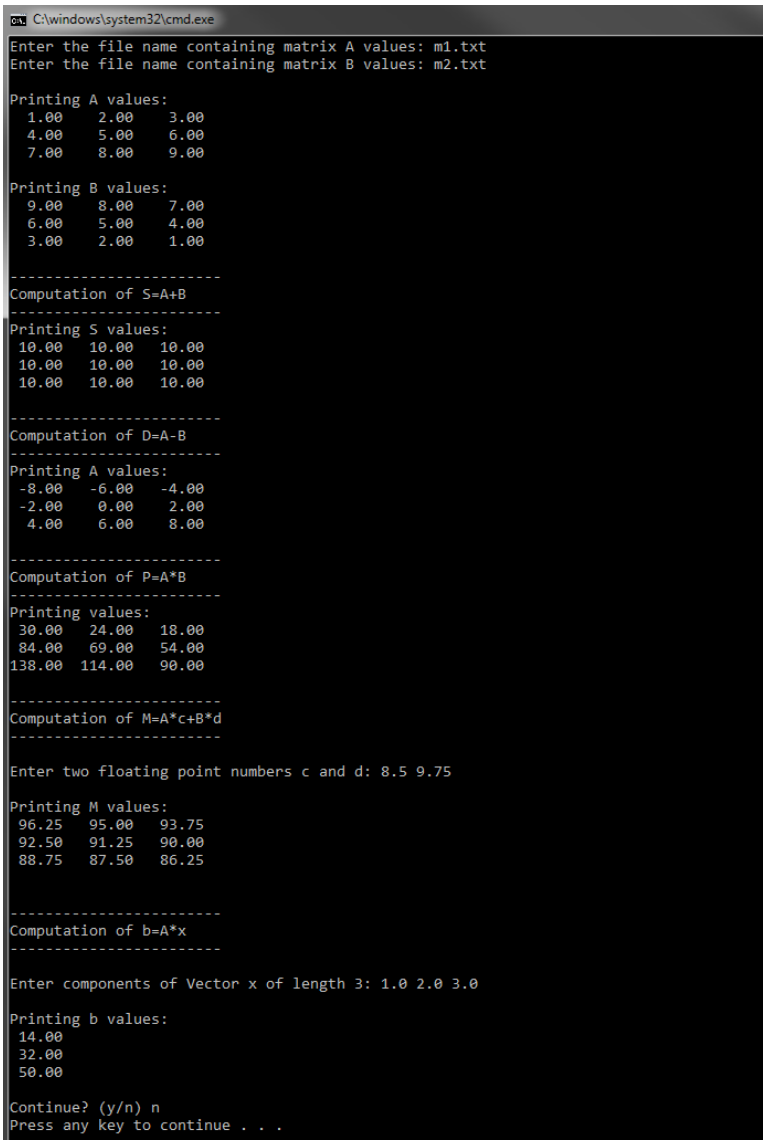
```
56          vector< vector<double> > values;  //2-d array storage of the matrix
57  };
```

- Once the class is implemented, one should be able to compile and run the following code:

```
int main(){
typedef Matrix::Vector Vector; //nickname for a Vector
using namespace std;
Matrix A(3, 3, 1.0); // 3 x 3 matrix with values 1.0
Matrix B(3, 3, 2.0); // 3 x 3 matrix with values 2.0
A(0,0) = B(2,2);
Matrix S = A+B;
Matrix D = A-B;
Matrix P = A*B;
Matrix M = A*3.0+B*4.0;
Vector x = { 1, 2, 3}; //vector of length 3
Vector y = A*x;
cout << A << B << S << D << P << M ;
cout << Matrix(y); // using anonymous Matrix constructed from y
}
```

- Write a program that reads values of two matrices from two separate files and then performs various calculations and prints them. For details, see the sample of input-output. Implement a loop in which the above actions are repeated until the user requests to quit.

- Sample input-output (see next page):



Remarks:

1. Assume that the matrix dimensions are always positive.
2. During the initialization of the `Matrix` object if either n or m, provided to the constructor with parameters, is 0, the vector `values` must stay empty.
3. Submit your solution as `hmw_7_2.cpp`.