

OOP - Midterm 1 – 2022.04.07

Exam Time: 18:30 ~ 21:20

There are four cases for scoring:

If you upload the file:

Before 04/07 21:20, your highest score will be 100.

04/07 21:20 ~ 23:59, your score will reduce 40 points. (highest score will be 60)

04/08 00:00 ~ 21:20, your score will reduce 60 points. (highest score will be 40)

After 04/08 21:20, your score will be 0.

There are two questions for this midterm. Try your best to complete all the questions.

Rules

1. Turn in:

- Make your directory as below

```

└─ OOP_mid1_studentID_YourName
   └─ Midterm_Q1
      ├── main_Q1.cpp
      ├── Matrix.cpp
      └─ Matrix.h
   └─ Midterm_Q2
      ├── sort.h
      ├── main_Q2.cpp
      └─ sort.cpp
```

- There should be your answer file (.cpp, .h) in each directory.

- Zip all files in one file and name it **OOP_mid1_studentID_YourName.zip**

2. The main.cpp file for each question:

It is only a basic testing for your code. We will test your code with other test data.

3. During the exam time, the only website you can access is new E3. (From 18:30~19:30) . You can download your lab codes or assignment codes from new E3 as reference.

4. You **must** use template to do this midterm exam, each question has its own template.

5. You **must** use Visual Studio C++ to do this midterm exam.

6. If your source code cannot be built, your score is 0.

7. If you cheat, your score is 0.

Q1. Matrix operation

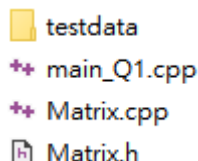
In this question, there are seven parts you need to complete.

You need to implement the following functions:

1. Operator Overloading : Implement operator overloading for addition and subtraction of matrices.
2. Matrix Add : Add two matrices if the shapes of them agree that they can.
3. Matrix Minus : Subtract two matrices if the shapes of them agree that they can.
4. Matrix Transpose :Transpose the matrix.
5. Matrix Multiply : Multiply two matrices if the shapes of them agree that they can.
6. Deep Copy : You need to copy the matrix and make the memory location of the two matrices different.
7. Print : You need to print a matrix and format it to our requirements.

Notice

1. If the shapes of two matrices do not agree that they can be calculated, Print the error message and return the first matrix parameter in that function.
2. Do not print messages that are not our request, except for your own debug log.
3. Follow the TODOs to implement your work. You **can not** edit all the other parts of the Q1 section.
4. You can add your own functions into the class, but make sure NOT to edit anything in the main function.
5. The test data should be put in the “testdata” folder like this:



```
testdata
++ main_Q1.cpp
++ Matrix.cpp
Matrix.h
```

Score Distribution(Total: 50 points)

Operator Overloading(+ , -) : each 5 points

Matrix Add : 5 points

Matrix Minus: 5 points

Matrix Transpose : 10 points

Matrix Multiply : 10 points

Deep Copy : 5 points

Print : 5 points

Input Format

We have two lines in one input file.

That is, we will give two matrices in one input file.

The first two numbers of each line represent the number of rows and the number of columns.

And the others are the values in the matrix.

Sample Input

3 2 2 3 5 7 11 13

2 1 17 19

Explanation

Matrix A:

```
┌  2  3  ┐
│  5  7  │
└ 11 13  ┘
```

Matrix B:

```
┌ 17  ┐
└ 19  ┘
```

Then, Transpose(A) will be:

```
┌ 2 5 11  ┐
└ 3 7 13  ┘
```

And Multiply(A, B) will be:

```
┌  91  ┐
│ 218  │
└ 434  ┘
```

P.S. You don't need to deal with problems of invalid matrix shapes, like Add(A, B) in this situation.

Sample Output

If your code is correct, it should output the result like this:

```
Print:
1 2 3
2 3 4

Construct: 6/ 6
Overloading: 4/ 4
    Add: 2/ 2
    Minus: 2/ 2
Transpose: 6/ 6
Multiply: 5/ 5
DeepCopy: 3/ 3
```

It will construct a (2 3 1 2 3 2 3 4) matrix and call your Print() first. Then read data in the input file automatically, and then call functions in Matrix.h and Matrix.cpp.

After that, it will compare the return data in your functions and check if it is correct. **If it's correct, it will print nothing.** But if your answer is wrong, it will show you a warning message.

Finally, it will calculate and show the number of questions you have answered correctly / Total questions.

For example : Add: 1/ 2

It means there are two ADD questions in total and you just answer one question correctly.

The wrong question will show on top like this:

```
While doing Add A with B, the answer should be:
2 2 2 2 3 5
But your answer is:
2 2 2 3 4 5
```

Q2. Sort and search

In this question, there are four parts you need to complete. The numbers are sorted in ascending order in this question.

You need to implement the following functions:

1. InsertionSort : use **insertion sort** to sort the numbers in the vector.
2. BucketSort : use **bucket sort** to sort the numbers in the vector, and you must print out the updated numbers in each pass. (**An algorithm of Bucket Sort is on the last page**)
3. BinarySearch : use **binary search** to check if the target number is in the vector.
4. Output: You need to print the result of the insertion sort and bucket sort.

Notice

1. Please implement the sort function **BY YOURSELF**. (Don't call any inbuilt sorting function!)
2. Follow the TODOs to implement your work. You **can not** edit all the other parts of the Q2 section.
3. In the first part, you must use **insertion sort** to sort, if you use the other sort algorithm, your score of this part will **be zero**.
4. In the second part, you must use **bucket sort** to sort, if you use the other sort algorithm, your score of this part will **be zero**.
5. In the third part, you must use **binary search** to search the number. If you use the other way to search, your score of this part will **be zero**.

Score Distribution(Total: 50 points)

Insertion Sort : 20 points

Bucket Sort : 10 points

Binary Search: 15 points

Output: 5 points

Input Format

Each two lines are one set. All the numbers are integers.

The first line will give you four numbers.

The first and the second numbers will only be **0** or **1**.

The second line will give some numbers, which mean that are data you need to sort.

	0	1
The first number at the first line	Use the insertion sort to sort the numbers at the second line	Use the bucket sort to sort the numbers at the second line
The second number at the first line	Don't execute the binary search	After sorting, execute the binary search to search the fourth number at the first line if it is in the vector.
The third number at the first line	The total number of the numbers at the second line.	
The fourth number at the first line	For the first line, if the second number is 0, then the fourth number will be 0. For the first line, if the second number is 1, then the fourth number is the target of the binary search.	

Output Format

For insertion sort, you need to output the result after sorting.

For bucket sort, you need to output the result each round.

For binary search, you need to output the result each step. At the last step, you need to print whether you find it. If you find it, print the target and "Found". Otherwise, print "Not Found".

Sample Input

(In midterm1_2.txt)

```
0 0 12 0
123 789 456 963 852 741 159 357 349 761 248 862
1 0 12 0
862 248 761 349 357 159 741 852 963 456 789 123
0 1 10 5
5 4 3 2 1 9 8 7 6 10
1 1 9 6
426 134 891 243 620 578 655 902 319
```

Sample Output

```
Insertion Sort : 123 159 248 349 357 456 741 761 789 852 862 963

Bucket Sort
1 : 741 761 852 862 123 963 456 357 248 159 349 789
2 : 123 248 349 741 159 357 456 852 761 862 963 789
3 : 123 159 248 349 357 456 741 761 789 852 862 963

Insertion Sort : 1 2 3 4 5 6 7 8 9 10
Searching : 5 Found

Bucket Sort
1 : 620 891 902 243 134 655 426 578 319
2 : 902 319 426 620 134 243 655 578 891
3 : 134 243 319 426 578 620 655 891 902
Searching : 578 243 134 Not found
```

Algorithm for Bucket Sort

Step 1: First find the largest element of the list and count its number of digits, and the number of digits is equal to your number of passes.

Step 2: Initialize buckets for all numbers from 0 to 9

Step 3: Consider a single digit first, then insert the numeric value into the corresponding bucket. For example, if the number of digits considered is 4, insert the key into the 4th bucket, and if it is 2, insert the numeric value into the 2nd bucket.

Step 4: Now sort each bucket.

Step 5: Put the items in the bucket back into the original sequence.

Step 6: Repeat steps (3) to (5) for all numbers you want to check.

Example for Bucket Sort

If the input array is {426, 134, 891, 243, 620, 578, 655, 902, 319}.

We found the maximum element (902) in this array and got its number of digits(3), so we just need to do three passes and create 10 buckets to store the numbers.

First pass(according to one digit)

Bucket	0	1	2	3	4	5	6	7	8	9
Number (Sorted)	620	891	902	243	134	655	426		578	319

This array will be updated to {620,891,902,243,134,655,426,578,319} after the first pass.

Second pass(according to tens digit)

Bucket	0	1	2	3	4	5	6	7	8	9
Number (Sorted)	902	319	426 620	134	243	655		578		891

This array will be updated to {902, 319, 426, 620, 134, 243, 655, 578, 891}

Third pass(according to hundreds digit)

Bucket	0	1	2	3	4	5	6	7	8	9
Number (Sorted)		134	243	319	426	578	620 655		891	902

Now, the sorted array is {134, 243, 319, 426, 578, 620, 655, 891, 902}