CIS 22B
Intermediate Programming Methodologies in C++
Programming Assignments

Homework 3

100 Points

Pointers and Dynamic Allocation of Memory

Project: Ragged Arrays (see next pages)

Grading

1.	main()	– 10
2.	Get data from file	– 20
3.	Sort each row in a ragged array // call Insertion Sort	– 15
4.	Sort the ragged array // Insertion Sort	– 20
5.	Write the ragged array	– 15
6.	Release (delete) the ragged array	– 10
7.	Report	– 10

Write a short report (not more than one page) to explain the design of your program:

→ Show how data are organized in a ragged array (draw)
// What are the arrays used in this program?

→ Show the structure of your program
// structure chart or pseudocode
// What are the functions used in this program

Run the program once and save the output at the end of the source file as a comment. Compress the source file, input and output files (if any), and the report, and upload the compressed file: 22B_LastName_FirstName_H3.zip

Project: Ragged Arrays

The program starts by reading data from a file into a dynamically allocated ragged array. The data file, **`ragged.txt`**, (see next page) begins with **`n`**, the number of rows. On the next **`n`** lines, for each row in the ragged array, there is an integer representing the size of that row, followed by the numbers (type double) on that row. Here is an example:

```
4
3 23.9 51.2 35.6
5 12.2 23.5 54.6 5.8 56.8
1 88.8
2 12.1 34.9
```

> A typical approach with ragged arrays is to allocate one extra row and place NULL as a sentinel value. This way, there is no need to pass the number of rows to other functions, such as sort or print, since you can use NULL to stop.

Call the insertion sort to sort each row in the ragged table in descending order:

```
3 51.2 35.6 23.9
5 56.8 54.6 23.5 12.2 5.8
1 88.8
2 34.9 12.1
```

> Modify the insertion sort function to 1. Sort in descending order, and 2. Replace int by double.

Sort the ragged table in descending order using a variation of the Insertion Sort algorithm to rearrange the rows from the longest to the smallest:

```
5 56.8 54.6 23.5 12.2 5.8
3 51.2 35.6 23.9
2 34.9 12.1
1 88.8
```

Write the ragged array to the screen as shown below. Format the ragged table providing a width of 2 for the size of the row. Show the floating-point numbers with one digit after the decimal point. Assume that the integer part of the number has up to three digits.

```
The ragged array has 4 rows:
 5  56.8  54.6  23.5  12.2   5.8
 3  51.2  35.6  23.9
 2  34.9  12.1
 1  88.8
```

Finally, release the memory and terminate the program.

Create the input file **`ragged.txt`**, with the following data:

```
13
3 23.9 51.2 35.6
5 12.2 23.5 54.6 5.8 56.8
1 88.8
2 12.1 34.9
4 23.9  3.7 51.2 35.6
6 12.2 23.5 888.8 54.6 10.8 56.8
2 88.8 0.5
3 12.1 111.5 34.9
3 3.5 5.1 5.6
11 1.2 3.5 1.6 0.8 6.2 7.5 2.1 1.2 9.0 8.9 5.3
7 99.9 12.2 23.5 888.8 54.6 10.8 56.8
2 2.9 384.5
5 25.2 38.4 4.6 125.6 6.3
```

EXTRA CREDIT: Replace all indices by pointers (see next page) and write a loop in main() to read, process, and write all the ragged arrays found in the input file. Create the input file **`ragged.txt`**, with the following data:

```
4
3 23.9 51.2 35.6
5 12.2 23.5 54.6 5.8 56.8
1 88.8
2 12.1 34.9
13
3 23.9 51.2 35.6
5 12.2 23.5 54.6 5.8 56.8
1 88.8
2 12.1 34.9
4 23.9  3.7 51.2 35.6
6 12.2 23.5 888.8 54.6 10.8 56.8
2 88.8 0.5
3 12.1 111.5 34.9
3 3.5 5.1 5.6
11 1.2 3.5 1.6 0.8 6.2 7.5 2.1 1.2 9.0 8.9 5.3
7 99.9 12.2 23.5 888.8 54.6 10.8 56.8
2 2.9 384.5
5 25.2 38.4 4.6 125.6 6.3
5
3 23.9 51.2 35.6
5 12.2 23.5 54.6 5.8 56.8
1 88.8
2 12.1 34.9
```

```
6 12.2 23.5 888.8 54.6 10.8 56.8
```

Three ways to display an array:

```
// A. Use an index
for( i = 0; i < size; i++ )
{
     cout << ary[i] << " ";
}


// B. Use an index and pointer arithmetic
// NEVER USE THIS STYLE!
for( i = 0; i < size; i++ )
{
     cout << *(ary + i) << " ";
}


// C. Use a pointer // ← This is the required style for the Extra Credit assignment
for( pW = ary, pLast = ary + size - 1; pW <= pLast; pW++ )
{
     cout << *pw << " ";
}
```