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

Homework 3

100 Points

Pointers and Dynamic Allocation of Memory

`22B_H3A_InsertSort.cpp` (pointers, arrays, and sorting)
`22B_H3B_Errors.cpp` (find and fix errors: dynamic memory allocation)
`22B_H3C_Ragged.cpp` **Project: Ragged Arrays** (see next pages)

Grading

Program 3A	– 10
Program 3B	– 20
Program 3C	
1. Get data from file	– 20
2. Show ragged table	– 10
3. Sort rows	– 10
4. Sort table	– 20
5. Release ragged table	– 5
Self Assessment Report	– 5

Self Assessment Report: Write a short report, (see `22B_H3Report.doc` form) briefly explaining your code and containing an assesment of your implementation based on the above grading criteria.

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

Next Page

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 (see Program A), 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
```

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 above. 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.

Finally, release the memory and terminate the program.

Next Page

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
```

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
for( pW = ary, pLast = ary + size - 1; pW <= pLast; pW++ )
{
    cout << *pw << " ";
}
```

Use style A to complete the program. If you wish to practice pointers, for a more challenging assignment, convert style A to style C. Never use style B!