In this application we will sort a list of complex numbers using the method involving pointers that was presented in the Lab 1 application. That is, we will rearrange the pointers instead of rearranging the complex numbers.

There is no natural order relation on the complex numbers, so on what basis will we sort a list of complex numbers? There are several possibilities.

• Sort lexicographically – Order the complex numbers primarily by their real parts. If the real parts are equal, then order them by their imaginary parts. For example, $2 + 5i < 4 + 3i$ because $2 < 4$. In this case, the 5 and the 3 do not matter. On the other hand, $2 + 5i > 2 + 3i$ because $2 = 2$ and $5 > 3$. Use the $\text{Re()}$ and $\text{Im()}$ member functions for this sort.

• Sort by absolute value – Order the complex numbers according to their absolute values. For example, $2 + 5i > 4 + 3i$ because $\sqrt{2^2 + 5^2} > \sqrt{4^2 + 3^2}$. Use the $\text{abs()}$ member function for this sort.

• Sort by amplitude – Order the complex numbers according to the angle they make with the positive x-axis. For example, $2 + 5i > 4 + 3i$ because $2 + 5i$ makes an angle of $(\approx 68^\circ)$ while $4 + 3i$ makes an angle of only $\approx 37^\circ$. Use the $\text{amp()}$ member function for this sort.

In this application, we want to use three lists of pointers, one for each sorting method, and then sort the complex numbers by all three methods. Rather than write three separate $\text{sort()}$ functions, which would all be identical except for the comparison statement, we will write one $\text{sort()}$ function and pass to it the comparison function that we wish to use. Let the three comparison function have prototypes

\[
\begin{align*}
\text{bool compareLex(Complx* p1, Complx* p2);} \\
\text{bool compareAbs(Complx* p1, Complx* p2);} \\
\text{bool compareAmp(Complx* p1, Complx* p2);} \\
\end{align*}
\]

Each of these functions will return $\text{true}$ if the first object pointed to is less than the second object pointed to.

Then the $\text{sort()}$ function will have prototype

\[
\text{void sort(Complx* p[], int size, bool(*comp)(Complx*, Complx*));}
\]

Note the description of the third parameter: $\text{bool(*comp)(Complx*, Complx*)}$. This means “a pointer to a function that takes two $\text{Complx}$ pointers as parameters and returns a $\text{bool}$.” That is exactly what our three comparison functions do. So, for example, to sort the complex numbers lexicographically, we would make the function call
sort(pLex, 10, compareLex);

where pLex is the name of the first array of Complx pointers and 10 is the size of the list. By passing the compareLex() function, the pointers will be sorted according to that logic. Similar function calls will sort by the other two methods.

Write a program named ComplexSorter.cpp that will read a list of complex numbers and then sort them by all three methods and then print the list three times, once for each method.

Place the files ComplexSorter.cpp, complx.h, and complx.cpp in a folder named Lab 02 and drag it to the dropbox.