SORTING

Today

- This lecture looks at a simple form of sorting.
- We illustrate this by sorting some numbers that have been read into an array.
- Before doing this, however, we will recap some old material by considering how to read data from a file.

Reading information from a file (again)

- Consider the following problem statement:

  Read six numbers from a file into an array.
- With what we have covered so far, we would probably write something like the code in `files.cpp` (which you can download from the class web page).
- First we set up a file as the input stream:

```
ifstream myfile;
myfile.open("numbers-short.txt");
```

- Then we use a `for` loop to read in the six numbers:

```
for(counter = 0; counter < 6; counter++)
  {
    myfile >> numbers[counter];
  }
```

- This is fine so long as we know that there are at least six numbers in the file.
- Sometimes we are not so lucky.
- We want to read in up to six numbers, but there may not be as many as that.

- In such a case (which is not uncommon) we might want to do the following:

```
do
  {
    myfile >> numbers[counter];
    counter++;
  }
while(counter < 6 && myfile);
```

- The second `myfile` is true so long as the last read from the file returned something.
- Thus when we get to the end of the file, `myfile` is false, and the loop ends.

- Another way to achieve the same thing is to do:

```
while(counter < 6 && !myfile.eof())
  {
    myfile >> numbers[counter];
    counter++;
  }
```

- The function `myfile.eof()` returns true if we are at the end of the file.
- Thus, once again, when we get to the end of the file, the loop terminates.

- You can test the way these work by running `files.cpp`, `files2.cpp` and `files3.cpp` from the course webpage.
- To test these, use the files of nubers `numbers.txt`, which holds more than 6 numbers, and `numbers-short.txt`, which holds less.
- Another useful function for handling files is `myfile.isopen()`, which will return false if a previous call to `myfile.open()` failed.
- Such a failure would occur, if you were opening a file for reading, if the file didn't exist (which is a problem that we have seen several times in the lab exercises).

Sorting

- Now we can read numbers into an array. Let's look at sorting them.
- We will start by looking at the *linear sort*, and consider sorting into increasing order (that is "smallest first").
- To linear sort, we look in turn at each member of the araay in turn.
- For each of these we look at all the memmers *later* in the array.
- If the later member is smaller, we swap the two.
- This algorithm translates quite simply into C++ using two nested `for` loops.

```
// Step through every position in the array in turn.
for(counter = 0; counter < 6; counter++)
  {
    // For each position, look through every later
    // position in the array.
    for(counter2 = counter; counter2 < 6; counter2++)
      {
        // If one of the later numbers is smaller, then swap
         if(numbers[counter2] < numbers[counter])
          {
            swap               = numbers[counter];
            numbers[counter]   = numbers[counter2];
            numbers[counter2]  = swap;
          }
      }
  }
```

Summary

- This lecture discussed two things.
- First it considered different ways of reading information in from a file.
  - We looked at a couple of ways of detecting the end of a file.
- Then we considered how to sort things.
  - In particular, we looked at the linear sort algorithm.