Arrays and Functions

- Things to note about C-style arrays:
  - An array is not a type
  - An array is a primitive C-style construct that consists of many items stored consecutively and accessed through a single variable name (and indexing)
  - This is actually done by remembering the starting address of an array, and computing an offset
  - The name of an array acts as a special kind of variable -- a **pointer** -- which stores the starting address of the array
- An array can be passed into a function as a parameter
  - Because an array is not a *single item*, the array contents are not passed "by value" as we are used to with normal variables
    - The normal meaning of "pass by value" is that the actual argument value is *copied* into a local formal parameter variable
    - In the case of arrays, just the *pointer* is copied as a parameter. We'll see this in more detail when we get to pointers
  - When an array is sent into a function, only its starting address is really sent
  - This means the function will **always** have access to the actual array sent in
  - Returning an array from a function works similarly, but we need pointers to use them well (not yet covered)
- Example function:

```
void PrintArray (int arr[], int size)
{
  for (int i = 0; i < size; i++)
    cout << arr[i] << ' ';
}
```

  Note that:
  - The varibale `arr` acts as the local array name inside the function
  - There is no number in the brackets. `int []` indicates that this is an array parameter, for an array of type `int`
  - It's usually a good idea to pass in the array size as well, as another parameter. This helps make a function work for any size array
- Sample call to the above function:

```
int list[5] = {2, 4, 6, 8, 10};

PrintArray(list, 5);          // will print: 2 4 6 8 10
```

Using `const` with array parameters

- Remember: When passing an array into a function, the function *will* have access to the contents of the original array!
- Some functions that *should* change the original array:
  - `Sort(), Reverse(), SwapElements()`
- What if there are functions that should *not* alter the array contents?

    - `PrintArray()`, `Sum()`, `Average()`
- Put `const` in front of the array parameter to guarantee that the array contents will not be changed by the function:

```
void PrintArray (const int arr[], const int size)
{
  for (int i = 0; i < size; i++)
    cout << arr[i] << ' ';
}
```

  Notice that the `const` on the variable `size` is a good idea in this example, too. Why?

arrayfunc.cpp -- Here's an example program illustrating a couple of functions that take array parameters.

Practice exercises

Try writing the following functions (each takes in one or more arrays as a parameter) for practice:

- A function called `Sum` that takes in an array of type `double` and returns the sum of its elements
- A function called `Average` that takes in an integer array and returns the average of it's elements (as a `double`)
- A function called `Reverse` that takes in an array (any type) and reverses its contents
- A function called `Sort` that takes in an array of integers and sorts its contents in ascending order
- A function called `Minimum` that takes in an array of type `double` and returns the smallest number in the array

---