CSE 232, Lab Exercise 11

The Problem

We are going to work on some dynamic memory with arrays. We are going to dynamically allocate an array by passing it to a function, then work with that array.

Your Tasks

I want you to add the needed functions in the provided main. Some comments about them.

- `size_t fill_from_file(long*&, string)`. A pointer reference (watch the videos) is passed, meaning that what the pointer can point to is allowed to change in the function. This is good, as we will allocate the memory in the function, and the pointer reference will point to that memory. Thus the pointer in the calling main program will also point to that memory. The size of array is determined by the contents of a file which you must open.
    - if the file does not exist, throw a runtime_error

  The format of the existing file is as follows:
    - every line has a single integer number
    - the first number in the file is the count of integers that follow
    - the remaining numbers are to be placed into to the passed array in the order in which they occur.
        - they all fit in a long.
    - you must allocate the memory **in the function** for that array as you cannot know the size required until you read the file.
        - the pointer reference should point to that dynamically allocated memory.
    - it returns the size of the dynamically allocated array.

- `void print_array(long*, size_t, ostream&)` Prints all the values in the array
    - first parameter is the array
    - second parameter is the size of the array.
    - third parameter is what stream
    - if you use iterators (or their equivalent), it is a single line.
    - no return

- `size_t concatenate(long*&, size_t, long*, size_t)`: concatenates the contents of the second array onto the end of the first array. First array's size will therefore change. Returns the size_t of the first array now that it has been expanded. Second array is not changed.
    - you have to allocate new memory in the function that is the size of the sum of the two sizes.
    - copy the contents of the first array into the new memory
    - copy the contents of the second array (starting at the end of the first) into the new memory
    - delete the first array
    - set the pointer of the first array to the new memory
        - as it is a pointer reference, the change will be reflected in the main.
    - return the new size

- `pair<long*, size_t> copy_evens(long ary[], size_t sz)` : Takes in an array, created previously, and looks for the even values in that array. It dynamically creates a new array big enough to hold the even values and you must copy the even values from the original array into this new array. You return a pointer to the new array and its size as a pair.

- the original array is the first parameter
- the size of the original array is the second parameter
- you must determine the size needed to hold the new array before you allocate it
- once you allocate it you can fill it with the even values from ary.
- you return a pair containing a pointer to the new array and its size.

Copy the main file provided into your code and provide the needed functions in that file. If you need functions besides the ones listed, go ahead and add them. Also, you must delete your dynamic memory in the main (see comment).

```
int main(){
    long *ary;
    long ary2[] ={10,11,12,13,14};
    size_t ary2_sz = 5;

    print_array(ary2, ary2_sz, cout);
    cout << endl;

    size_t sz_file = fill_from_file(ary, "tables.txt");
    print_array(ary, sz_file, cout);
    cout << endl;

    size_t sz_concat = concatenate(ary, sz_file, ary2, ary2_sz);
    print_array(ary, sz_concat, cout);
    cout << endl;

    pair<long*, size_t> p = copy_evens(ary, sz_concat);
    print_array(p.first, p.second, cout);
    cout << endl;
    // add code to delete dynamic memory after this
    delete [] ary;
    delete [] p.first;
}
```

Helpful C++ generic functions (look them up on the web)

- `count_if`
- `copy_if`

What does a parameter declared with the type `long *&` mean? It means that what is being passed is a reference value for a pointer. If you change what the pointer points to in the function, that change will be reflected in the calling main.

Testing

Provided is a `lab11_arrays.h` and the example main `lab11_main.cpp`. You must write `lab11_arrays.cpp`. Turn it into Mimir as a director, `lab11/lab11_arrays.cpp`

If you get an error about function missing, did you remember to compile both the main and the functions together as a single executable:

```
g++ -std=c++11 –Wall lab11_main.cpp lab11_arrays.cpp
```