Assignment #6

Due: Friday, July 18

Objective

To gain experience with bitwise operations, used inside a class. Also will provide further practice with dynamic allocation.

Task

You will implement a class called `BitArray`, which will simulate a list of bits of any size, which can be individually set, cleared, flipped, and queried. You will also implement a function that is to be used by a sample test program, which uses the Sieve of Eratosthenes technique (with the bit array) to find prime numbers.

Details

Download these starter files:

- `bitarray.h` -- declaration of the BitArray class, listing out its primary interface (public functions)
- `main.cpp` -- a sample program for finding prime numbers

The `BitArray` class

Implement the `BitArray` class, defining all specified public member functions, in the file **`bitarray.cpp`**. Here are some details about the BitArray class:

1. A bit array is to be implemented with an underlying array of type `unsigned char`. 'Unsigned' because we are only interested in bits, not in negatives, and type `char` because it is the smallest integer type. The concept of a `BitArray` object is that it will store an array of bits (in the smallest storage space needed), indexed starting at 0, just like with normal arrays.

2. The array of characters should be created dynamically. The primary constructor has one parameter, which indicates how many **bits** are needed. The constructor should allocate the *minimum* number of cells needed for this many bits.
   Also, have the constructor initialize all bits to 0. Example:

```
    BitArray xy(35);      // builds storage for at least 35 bits
                          //  if we assume 1 byte char, this takes 5
                          //  characters, for a total of 40 bits
```

3. The `Length()` function should return the total number of bits in the allocated array. In the example above (assuming 1 byte char), this is 40

4. While type `char` is commonly 8 bits on most machines today, you may *not* assume that this is always the case. Structure your class so that it is versatile enough to handle different platforms (where type `char` might differ in size). But always use the minimum number of `char` elements when creating the array. Hint: `sizeof()` is a function call that returns the exact number of bytes taken by a variable or type on a given machine:

   ```
   int size = sizeof(int);       // tells how many bytes for an int
                                 //  on current machine
   ```

   Suggestion: Use a constant to store the size of an unsigned char in the program, for modifiable computations later. If using only inside the class, a static const is best.

5. Because dynamic allocation is used, the BitArray class should implement an appropriate destructor, copy constructor, and assignment operator (for deep copy and appropriate cleanup)

6. The functions `Set(), Unset(), Flip(), and Query()` represent the different things that can be done with one bit. Each function takes in an index number -- the index of the bit in question.
   - `Set()` should set that bit to 1, without affecting any others
   - `Unset()` should set that bit to 0, without affecting any others
   - `Flip()` should change that bit to its opposite, without affecting any others
   - `Query()` should return `true` if that bit is currently 1, and it should return `false` if that bit is currently 0

7. The operator overloads:
   - `operator<<` -- the insertion operator should be written to do output of a BitArray object. Format is the entire array, printed as one continuous sequence of bits, inside parintheses. See example outputs from test program
   - `operator==` and `operator!=` -- usual inequality operators. Entire arrays must match for them to be equal

8. General:
   - You may add private functions to the class if you like, and you may add private constants. You may not change the public interface or the underlying storage (dynamic array of unsigned char).
   - Note that NOT ALL features of the BitArray class are tested in the provided `main.cpp` sample program. It is up to you to test all BitArray features.

Sieve of Eratosthenes

A common algorithm to find prime numbers is the Sieve of Eratosthenes. The `main.cpp` program provided already sets up a BitArray object of desired size. Then it calls upon a function named `Sieve`.

Write the `Sieve()` function in a file called `sieve.h`. Do not change `main.cpp` in any way. The `Sieve()` function should follow the Sieve of Eratosthenes pattern. The general algorithm is as follows:

1. Start by initializing all bits in the array to 1.
2. Each index of the bit array will represent one non-negative integer. Your algorithm should mark all **non-prime** numbers by setting these bits back to 0, proceeding as follows:
   - 0 and 1 are never prime. Unset these bits to 0

    - The next "uncleared" bit is prime. Leave this bit as a 1, but change all *multiples* of this value (not counting itself) to 0
    - Move to the next "uncleared" bit and repeat
    - This process only needs to repeat up to the square root of the array's length. (Example: If we are checking for the prime numbers from 0 through 500, then we can stop when we've reached sqrt(500), which is 22.36. Once we've reached an "uncleared" bit that is 23 or more, we know we've cleared all the non-primes
3. The remaining bits (which are still 1) indicate the primes.

You can find the `sqrt()` (square root) function in the library `<cmath>`.

Sample Runs

These are sample runs of the main.cpp program, the Sieve program to find primes. Remember to write your own driver(s) to test other functions in class `BitArray` (such as comparison operators, copy constructor, etc).

Note that in the sample runs, the bit array really is printing on one line -- but it will probably show on screen wrapped around to multiple lines

Sample run 1

```
Enter a positive integer for the maximum value: 345
The bit array looks like this:
(0011010100010100010100010000010100000100010100010000010000010100000100010100000
10001000001000000010001010001010001000000000000010001000001010000000001010000010
00001000100000100000101000000000101000101000000000001000000000001000101000100000
10100000000010000010000010000010100000100010100000000010000000000000100010100010
000000000000100000100000000010100)

Primes less than 345:
2       3       5       7       11      13      17      19
23      29      31      37      41      43      47      53
59      61      67      71      73      79      83      89
97      101     103     107     109     113     127     131
137     139     149     151     157     163     167     173
179     181     191     193     197     199     211     223
227     229     233     239     241     251     257     263
269     271     277     281     283     293     307     311
313     317     331     337
Goodbye!
```

Sample run 2

```
Enter a positive integer for the maximum value: 800
The bit array looks like this:
(0011010100010100010100010000010100000100010100010000010000010100000100010100000
10001000001000000010001010001010001000000000000010001000001010000000001010000010
00001000100000100000101000000000101000101000000000001000000000001000101000100000
10100000000010000010000010000010100000100010100000000010000000000000100010100010
00000000000010000010000000001010001000001000000010000010000010001000001000000010
00100000001000000000101000000000101000001000100000100000001000101000100000000000
10000000100010000000100010000010000000000010100000000000000000100000100000000010
00001000001010000010000000001000001000001010000010000010001010000000000010000000
00101000100000100000101000000000001000100000100000001000000000100000001000000000
10000000100000100000100010000000100000100010000000100010000000000000100000000010
0)

Primes less than 800:
2       3       5       7       11      13      17      19
23      29      31      37      41      43      47      53
59      61      67      71      73      79      83      89
97      101     103     107     109     113     127     131
137     139     149     151     157     163     167     173
179     181     191     193     197     199     211     223
227     229     233     239     241     251     257     263
269     271     277     281     283     293     307     311
313     317     331     337     347     349     353     359
367     373     379     383     389     397     401     409
419     421     431     433     439     443     449     457
461     463     467     479     487     491     499     503
509     521     523     541     547     557     563     569
571     577     587     593     599     601     607     613
617     619     631     641     643     647     653     659
661     673     677     683     691     701     709     719
727     733     739     743     751     757     761     769
773     787     797
Goodbye!
```

Submitting

Email these files:

```
  bitarray.h
  bitarray.cpp
  sieve.h
```

To finlayso@cs.fsu.edu