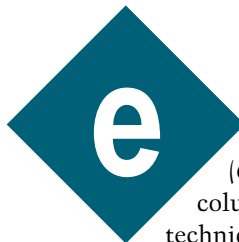


## LESSONS FROM THE TRENCHES

George Martin

### Let's Play a Game

George changes gears this time around as he trades problem solving for PC fun. In this article, he reminds us about the old sieve of Eratosthenes and goes about finding prime numbers using this technique as compared to a more modern approach. All is meant to be an effort of enjoyment, so put the tension aside and see what you can do with the power of computing.



very month, we (*Circuit Cellar* columnists) write our technical articles in

hopes of helping you get to the bottom of certain problems. Well, for a change, I'm not going to do any problem solving this month. I'm just going to have a little fun with the PC.

Most of us have moved up to a Pentium by now. I have a 200-MHz PII from Intel and a 233-MHz from AMD. I've got several '386 and '486 machines running Win95 that I use in circuit emulators and test machines. Win95 and the latest software packages claim 32-bit capabilities, and for a long time, I just thought 32 bits was twice as big as 16 bits and never gave it more thought. But, think about it for a minute.

Sixteen-bit numbers can go from 0 to 65,536, but 32-bit numbers go from 0 to 4,294,967,296. That's a big difference. These machines are capable of doing integer arithmetic on that 32-bit number in one clock cycle. And, the

move to 64-bit numbers (1.844674407371e + 19) is just a double precision step away. That's 18,446,744,073,709,551,616, give or take a million.

#### REMEMBER THE SIEVE

What can we do with all this computing power? Remember the sieve of Eratosthenes. Years ago, the sieve was used as a benchmark for CPU and compiler performance. I even remember articles about how compiler vendors were tuning their compilers to shine when the sieve was tested.

In case you don't remember, the sieve of Eratosthenes is a technique for finding prime numbers. The technique starts with a table of all the numbers you want to test. The table size is N, and the first sieve point is two. You mark each number after two that is a multiple of two, look for the first unmarked number after two, and repeat the process. (Check out some of the web references listed at the end of this article.)

This is meant to be a fun exercise not a hair-pulling nail-biting expedition, so I started my search for primes with an array. Every time I found a prime, I entered it into the array. I've limited its size to a reasonable number for now.

My first attempt was *Prime3.exe* and *PrimeMain3.c*, and they are as follows:

```
#define MAX_PRIME 80000
UINT32PrimeFound[MAX_PRIME];
for (j=0; j< MAX_PRIME; j++) {
    PrimeFound[j] = 0;
    // Zero out the table
}
```

You can start with the numbers 1, 2, and 3, which are all primes. The printf statements go to the console, which is a MSDOS window (see Listing 1).

## TESTING 2, 4...

Now, let's talk about the test. If you have a candidate for a prime number, you need to test against all numbers from two to the square root of that candidate. The square root when squared would be the candidate, so any larger number would square to larger than that candidate. Also, there is an integer divide in the CPU, so let's look at the C-language's percent operator and see if we get a medal or get buried (see Listing 2).

Listing 2 tests all the prime numbers found as divisors for the candidate. If any divide is without a remainder, then the candidate is not a prime. The testing for primes starts at the number five and goes to *LAST\_NUMBER*, another constant that you can alter for testing.

There is a square root function but only one for each candidate. It costs more but saves a lot in testing. You'll also notice a variable *LastPrime*. I did that for safety, but I'm not sure if it's needed. If this works, I'll investigate further.

The code for the percent operator is compiled as:

```
66: if (i%PrimeFound[j] == 0){
/* % ismod is divide and return remainder */
00401138 8B 4D F0mov ecx,dword ptr [j]
0040113B 8B 45 F4mov eax,dword ptr [i]
0040113E 33 D2 xor edx,edx
00401140 F7 34 8D A0 B6 41 00div
eax,dword ptr [ecx*4+41B6A0h]
00401147 85 D2 test edx,edx
00401149 75 09 jnemain
(0x00401154)+144h
```

All in all, this is not so bad. A couple of moves—an *xor* (which zeros out the high part of the 64-bit divisor), the *div* instruction, and a *test*.

This program can run in a DOS window and pipe its output to a file. That file would then be the primes that have been found. I did that for the program I've discussed so far and got a file with the last entries as follows:

```
78497 999979
78498 999983
```

So, 999,979 and 999,983 are prime numbers. And, there are 78,498 prime

Listing 1—Here you can see that the *printf* statements go to the console.

```
printf("Prime Number Locator....V00-03\n");
// Version Number That's Good!!
printf("Found the following primes.\n");
PrimeFound[0] = 1; /* One is a prime ...Trivial... */
LastPrime = 0;
printf("%d %d\n",LastPrime,PrimeFound[LastPrime]);
/* Output it */

PrimeFound[1] = 2; /* Two is a prime */
LastPrime = 1;
printf("%d %d\n",LastPrime,PrimeFound[LastPrime]);
/* Output it */

PrimeFound[2] = 3; /* Three is a prime */
LastPrime = 2;
printf("%d %d\n",LastPrime,PrimeFound[LastPrime]);
/* Output it */
```

Listing 2—The C-language's percent operator can be seen here.

```
#define LAST_NUMBER 1000000
for (i = 5; i < LAST_NUMBER; i=i+2) {
/* test all odd numbers */
FoundOne = 1; /* Set a flag */
for (j = 2; (j < LastPrime) || (i < Limit); j++) {
/* see if any previous prime is a factor */
if (i%PrimeFound[j] == 0) {
/* % is mod is divide and return remainder */
FoundOne = 0; /* It's not a prime */
break;
}
if (FoundOne == 1) { /* This is a prime */
PrimeFound[++LastPrime] = i; /* Save it */
printf("%d %d\n",LastPrime,PrimeFound[LastPrime]);
/* Output it */
}
}
```

numbers from 1 to 1,000,000. (In fact, one of the web references confirms that result.) However, this program takes a while to run. I can't be doing this all day; I've got work to do.

## TRY, TRY AGAIN

My second attempt was *Prime4.exe* and *PrimeMain4.c*. This technique of looking for prime numbers deviates from the classic sieve of Eratosthenes. But, let's begin with that classic approach.

I started with bit arrays. These arrays are 32 bits wide and INT32

deep. Again, I set limits to the array sizes for practicality and testing. This implementation uses these bit arrays to mark off numbers that are not prime, leaving only prime numbers.

First, fill the array with only odd numbers because even numbers aren't prime numbers:

```
for (j = 0; j < MAX_PRIME; j++) {
PrimeArray[j] = 0xaaaaaaaa;
// Fill the array with starting values
}
```

Then, prime out the first primes, 1 and 2, and start the loop. The first bit

you find set is a prime. Print it out, mark off all the multiples of that bit, and then keep looping and reporting.

This routine starts slowly because there are lots of numbers to mark off when the primes found are small, but after it starts, the loops get faster and faster with each pass. I've run these programs looking for primes up to 1,000,000. The results compare to the first attempt. The game is then to find all the primes up to 4,294,967,296. Because there are 32 bits packed into the array, you could easily go to  $32 \times 4,294,967,296$ , or 137,438,953,472. Good luck! ☒

*George started his career in the aerospace industry in 1969. After five years at a real job, he set out on his own and cofounded a design and manufacturing firm. Typical systems that George designs include servo-motion control, graphical input and output, data acquisition, and remote control. George is a charter member of the Ciarcia Design Works Team and most recently, he's been working on the people-tracking system for Bill Gates' new house. You can reach him at [george.martin@worldnet.att.net](mailto:george.martin@worldnet.att.net).*

## RESOURCES

### Web articles for the sieve of Eratosthenes

<http://www.mcn.net/~jimloy/sieve.html>

<http://hissa.nist.gov/dads/HTML/sieve.html>

<http://mathworld.wolfram.com/EratosthenesSieve.html>

[http://www.alliance.unm.edu/documentation/PGI/pgC++\\_lib/stdlibug/exa\\_7254.htm](http://www.alliance.unm.edu/documentation/PGI/pgC++_lib/stdlibug/exa_7254.htm)

[http://www.uni-bielefeld.de/~achim/prime\\_sieve.html](http://www.uni-bielefeld.de/~achim/prime_sieve.html)

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, [subscribe@circuitcellar.com](mailto:subscribe@circuitcellar.com) or [www.circuitcellar.com/subscribe.htm](http://www.circuitcellar.com/subscribe.htm).