

Forum:[Hack Forums](#)

Thread: [Primes with the Sieve of Eratosthenes](#)

JJaeger - October 7, 2011, 1:56 pm

Hey all!

I was bored today so I wrote a function that calculates all prime numbers between 2 and n. So I looked up a few ways to do it. There are quite alot options to calculate primes but I ended up with the [Sieve of Eratosthenes](#).

This option might not be the most efficient nor the easiest but it was fun to make. On my laptop it calculated the prime numbers between 0 and 100,000 in 5,042 milliseconds. I intentionally didn't use a STL container. If you got any comments just post them below. I hope someone can learn from this.

code:

Code #include <iostream>

/*

calculates prime numbers between 0 and n.

the amount of primes calculated will be assigned to primes, so you can loop through the array

returns a pointer to an dynamically allocated int array

*/

int* sieve(int n, int& primes)

{

int *arr = new int[n];

//fill array

for(int i = 0; i < n; ++i)

{

arr[i] = i;

}

//By definition 0 and 1 are not prime

arr[0] = -1;

arr[1] = -1;

//set every number that's not prime to -1

int ps[4] = {2, 3, 5, 7};

for(int i = 0, p = ps[0]; i < 4; ++i, p = ps[i])

{

for(int i = p + p; i < n; i += p)

{

arr[i] = -1;

}

}

//assign every prime number to the beginning of the array

```

int index = 0;
for(int i = 0; i < n; ++i)
{
    if(arr[i] != -1)
        arr[index++] = arr[i];
}

primes = index; //index contains the amount of prime numbers calculated now

//free the remaining unused memory
int *ptr = &arr[index];
delete [] ptr;

//return the array containing the prime numbers
return arr;
}

int main()
{
    int sz;
    int *arr = sieve(300, sz);

    //print out prime numbers
    for(int i = 0; i < sz; ++i)
        std::cout << arr[i] << std::endl;

    delete [] arr;
    return 0;
}

```

ohlawd - October 7, 2011, 2:02 pm

You should change it to use `std::vector` instead of dynamically creating an array, you could then remove each non-prime number rather than setting it to -1.

edit:

(Oct 7, 2011 01:56 PM)JJJaeger Wrote: [\[CITE\]](#) I intentionally didn't use a STL container. Sorry, didn't see this...

JJaeger - October 7, 2011, 2:13 pm

I'm not sure but I think my way is faster then removing non prime numbers from vectors. Every time you remove one element it needs to shift the elements on the right to the left. I guess that will take a sh*tload of time.

I also don't feel like writing one with vectors today but I got this code first:

```

Code #include <iostream>
    using std::cout;
    using std::endl;
#include <vector>
    using std::vector;
#include <ctime>

vector<int>& sieve(int n)

```

```

{
    int *arr = new int[n];

    for(int i = 0; i < n; ++i) //fill in array!
    {
        arr[i] = i;
    }

    arr[0] = -1;
    arr[1] = -1; //0 and 1 aren't prime by definition

    int ps[4] = {2, 3, 5, 7};
    for(int i = 0, p = ps[0]; i < 4; ++i, p = ps[i])
    {
        for(int i = p + p; i < n; i += p)
        {
            arr[i] = -1;
        }
    }

    vector<int> *primes = new vector<int>;
    for(int i = 0; i < n; ++i)
    {
        if(arr[i] != -1)
            primes->push_back(arr[i]);
    }
    return *primes;
}

int main()
{
    clock_t start = clock();

    vector<int> primes = sieve(100);

    for(vector<int>::iterator it = primes.begin(); it != primes.end(); ++it)
    {
        cout << *it << endl;
    }

    float time = (clock() - start);
    time /= 1000;

    cout << "Time elapsed: " << time << " seconds!\n";
}

```

It's ~150 ms slower on my laptop.

ohlawd - October 7, 2011, 2:22 pm

(Oct 7, 2011 02:13 PM)JJJaeger Wrote: [CITE](#) I'm not sure but I think my way is faster then removing non prime numbers from vectors. Every time you remove one element it needs to shift the elements on the right to the left. I guess that will take a sh*tload of time.

I also don't feel like writing one with vectors today.

Yeah, `std::list` would actually be better, I've read you should use `vector`/`list` whenever possible instead of dynamically creating arrays in C++. Just safer I guess, oh well.?

edit: ^ that doesn't look right :P, you should be able to get it faster than that.

JJaeger - October 7, 2011, 2:40 pm

Ye i know a STL container would be a better option but sometimes I'm in the mood to use dynamically allocated arrays. In a real project I probably wouldn't have done it like this :p

ohlawd - October 7, 2011, 2:49 pm

Well if you care, here is a nicer way to do it imo.

```
Code #include <iostream>
#include <vector>
#include <math.h>

using std::cout;
using std::cin;
using std::endl;
using std::vector;

bool IsPrime(double num)
{
    bool isprime = false;
    for(int i = 2; i <= sqrt(num); i += 2)
    {
        if(i % 2 == 0)
            i++;

        if((int(num)% i) == 0)
        {
            isprime = true;
            break;
        }
    }

    return isprime;
}

void sieve(vector<int>& primeNumbers, int num)
{
    for (int i = 0; i < num; ++i)
    {
        if(IsPrime(i))
        {
            primeNumbers.push_back(i);
        }
    }
}

int main()
{
    vector<int> primes;
```

```

sieve(primes, 300);

//print out prime numbers
for (vector<int>::iterator itr = primes.begin(); itr < primes.end(); ++itr)
{
    cout << *itr << endl;
}

cin.ignore();
cin.get();

return 0;
}

```

tflo - October 7, 2011, 4:22 pm

(Oct 7, 2011 02:13 PM)JJJaeger Wrote: [\[CITE\]](#) [code]#include <iostream>
using std::cout;
using std::endl;
#include <vector>
using std::vector;
#include <ctime>

I like your style of importing the scope right after including the header, might have to jack that. ?

Navigator - October 8, 2011, 1:35 am

JJaeger, your code is quite messy.

Shodan, you're doing trial division and using that for sieving is very slow, additionally your IsPrime function is totally flawed. Also in order for Sieve of Eratosthenes to work you can't remove elements while the algorithm is working - if you find a prime you need to increment the array index by that prime to eliminate all multiples.

Here's a clean way of doing it.

Code #include <iostream>
#include <vector>

```

std::vector<unsigned int> primeSieve(unsigned int n);

```

```

int main(void)
{
    std::vector<unsigned int> primes = primeSieve(10);

    for (std::vector<unsigned int>::const_iterator i = primes.begin(); i != primes.end(); i++)
    {
        std::cout << *i << std::endl;
    }

    std::cin.sync();
    std::cin.get();
    return 0;
}

```

```

std::vector<unsigned int> primeSieve(unsigned int n)

```

```

{
    std::vector<unsigned int> primes;
    unsigned int* buffer = new unsigned int[n];
    buffer[0] = 0;
    buffer[1] = 0;

    for (unsigned int i = 2; i < n; i++)
    {
        buffer[i] = i;
    }

    for (unsigned int i = 0; i < n; i++)
    {
        unsigned int current = buffer[i];

        if (current != 0)
        {
            primes.push_back(current);

            for (unsigned int j = i + current; j < n; j += current)
            {
                buffer[j] = 0;
            }
        }
    }

    delete [] buffer; buffer = nullptr;

    return primes;
}

```

JJaeger - October 8, 2011, 5:50 am

@Navigator, as stated in my first post I intentionally didn't use the STL. This because I like to do it on my own. In a real project I would've used the STL. If you got any cleaner solution without using the STL I'd like to see it.

Thanks for your feedback, though.

@Shodan, Navigator is right. Your code doesn't produce prime numbers.