University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 220: Computer Systems & Programming

Programming with Functions

Stop Decomposing at the Level of C Functions

You have seen several **C** functions.

As you gain more experience,
- when you break down tasks,
- **stop when a subproblem**
- **can be implemented with a function**,
- including an API call* to a library.

*API stands for Application Programming Interface, the functions that a library provides.

Let's Print Prime Numbers as an Example

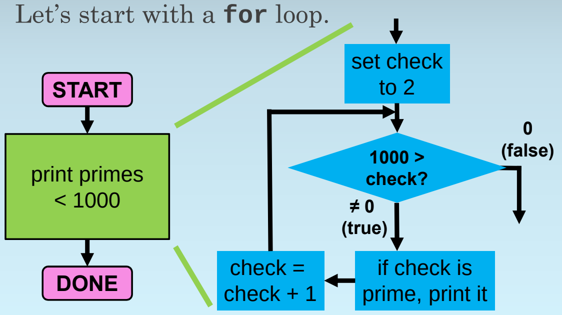
Let's solve this task with such an approach:

Print all prime numbers less than 1000.

In **MP5**, you will develop
- a body of drawing subroutines
- that build on
  - setting pen color and
  - drawing a dot.

Let's Decompose the Problem

Let's start with a `for` loop.



Stop When We Can Write as C Functions

We can check for primality with a C function!

But we need a function signature…

```
int32_t is_prime (int32_t num);
// Returns 1 if num is prime,
// or 0 if num is not prime.
```

Now we're ready to write `main`.

Our `main` Function for Printing Primes < 1000

```
int main ()
{
    int32_t check;
    for (check = 2; 1000 > check;
         check++) {
        if (is_prime (check)) {
            printf ("%d is prime.\n",
                    check);
        }
    }
    return 0;  // success, by convention
}
```

Remind Me: What Does Prime Mean Again?

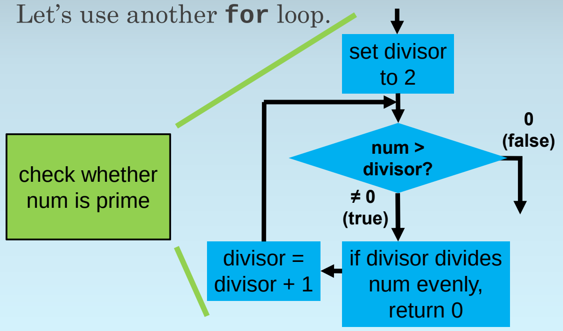
Let's write `is_prime`.

When is a number N prime?

N is prime iff
- **only 1 and N**
- **divide the number evenly.**
- **(N is not a multiple of anything but 1 and N.)**

Let's Decompose the Problem

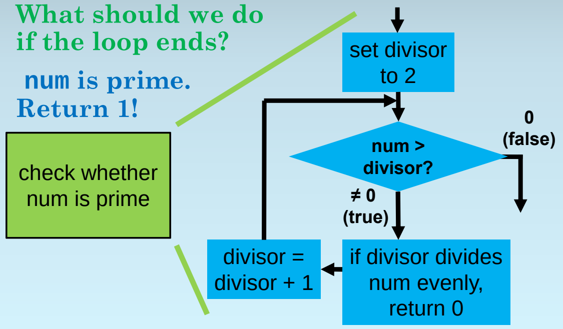
Let's use another `for` loop.



What's True if the Loop Ends?

What should we do if the loop ends?

`num` is prime. Return 1!



Stop When We Can Write as C Functions

We can check whether numbers divide evenly with a C function!

But we need a function signature…

```
int32_t divides_evenly
    (int32_t divisor, int32_t value);
// Returns 1 if divisor divides
// value evenly, or 0 otherwise.
```

Now we're ready to write `is_prime`.

Our `is_prime` Function for Checking Primality

```
int32_t is_prime (int32_t num)
{
    int32_t divisor;
    for (divisor = 2; num > divisor;
         divisor++) {
        if (divides_evenly
            (divisor, num)) {
            return 0;
        }
    }
    return 1;
}
```

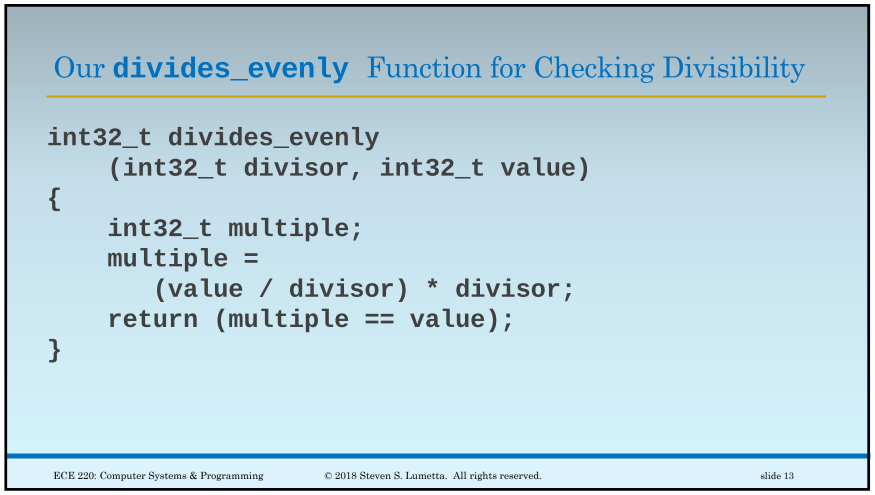
Use Integer Arithmetic to Test for Multiples

For integers **A** and **B**, what does the expression **(A / B) * B** produce?

The largest multiple of B that is not more than A.

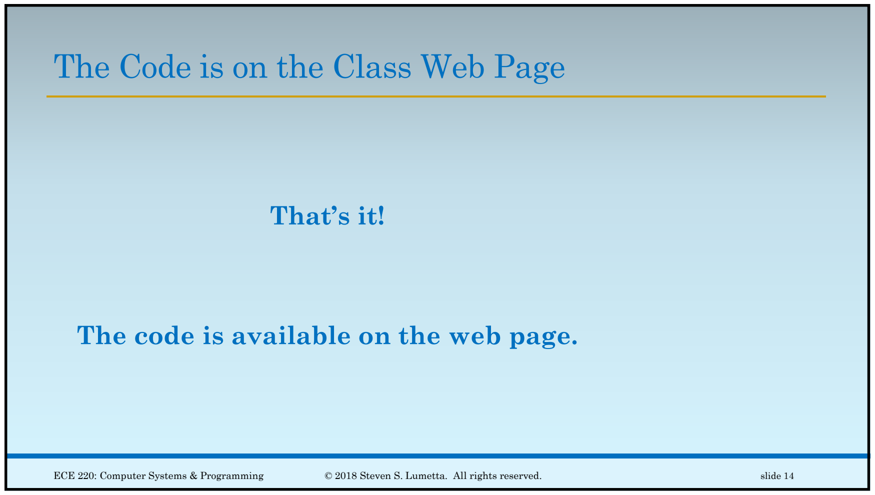
Let's use this expression to write **`divides_evenly`**.*

*Equivalently, one can just use modulus.



Our **divides_evenly** Function for Checking Divisibility

```
int32_t divides_evenly
    (int32_t divisor, int32_t value)
{
    int32_t multiple;
    multiple =
       (value / divisor) * divisor;
    return (multiple == value);
}
```



The Code is on the Class Web Page

That's it!

The code is available on the web page.