Chapter 20:

Recursion

Introduction to Recursion

- A *recursive function* contains a call to itself:

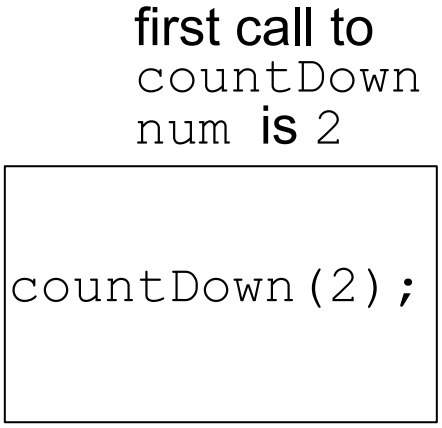
```
void countDown(int num)
{
    if (num == 0)
        cout << "Blastoff!";
    else
    {
        cout << num << "...\n";

        // recursive call
        countDown(num-1);
    }
}
```

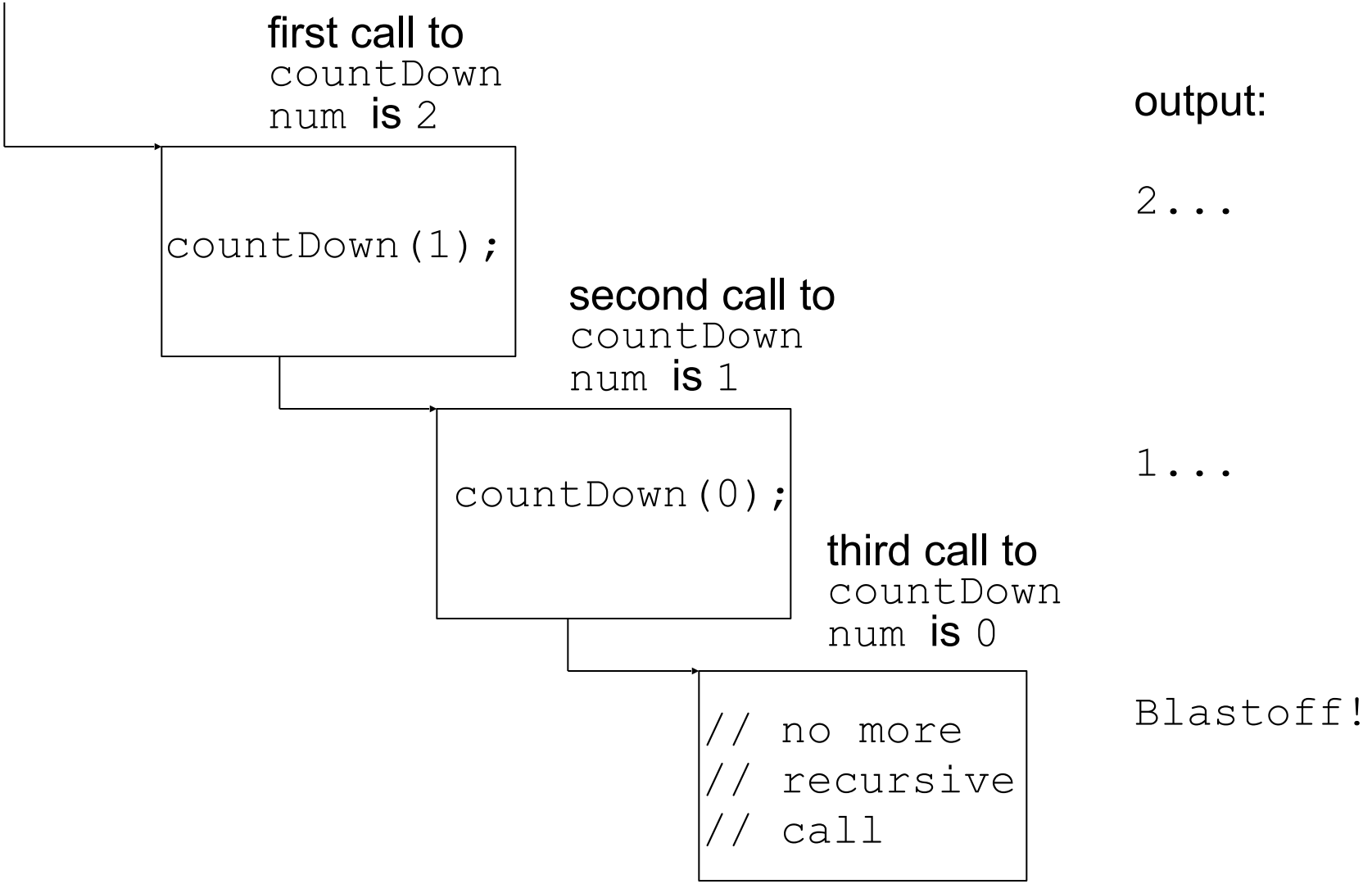
What Happens When Called?

Let us examine when a program contains a line like

```
countDown(2)
```



What Happens When Called?



Displaying messages with recursion

```
void message(int times)
{  // 1: anchor: loop terminating condition
   if(times <=0) return;

   // 2: body of the recursion: do something with the value
   cout << "This is a recursive function.\n";

   // 3: recursion call: update and repeat
   message(times - 1);
}
```

20.2

Solving Problems with Recursion

Recursive Functions – Purpose

- Recursive functions are used to reduce a complex problem to a simpler-to-solve problem.
- The simpler-to-solve problem is known as the *base case*
- Recursive calls stop when the base case is reached

Stopping the Recursion

- A recursive function must always include a test to determine if another recursive call should be made, or if the recursion should stop with this call
- In the sample program, the test is:

```
if (num == 0)
```

Stopping the Recursion

```
void countDown(int num)
{
   if (num == 0) // test
         cout << "Blastoff!";
   else
   {
        cout << num << "...\n";
        countDown(num-1); // recursive
   }                      // call
}
```

Stopping the Recursion

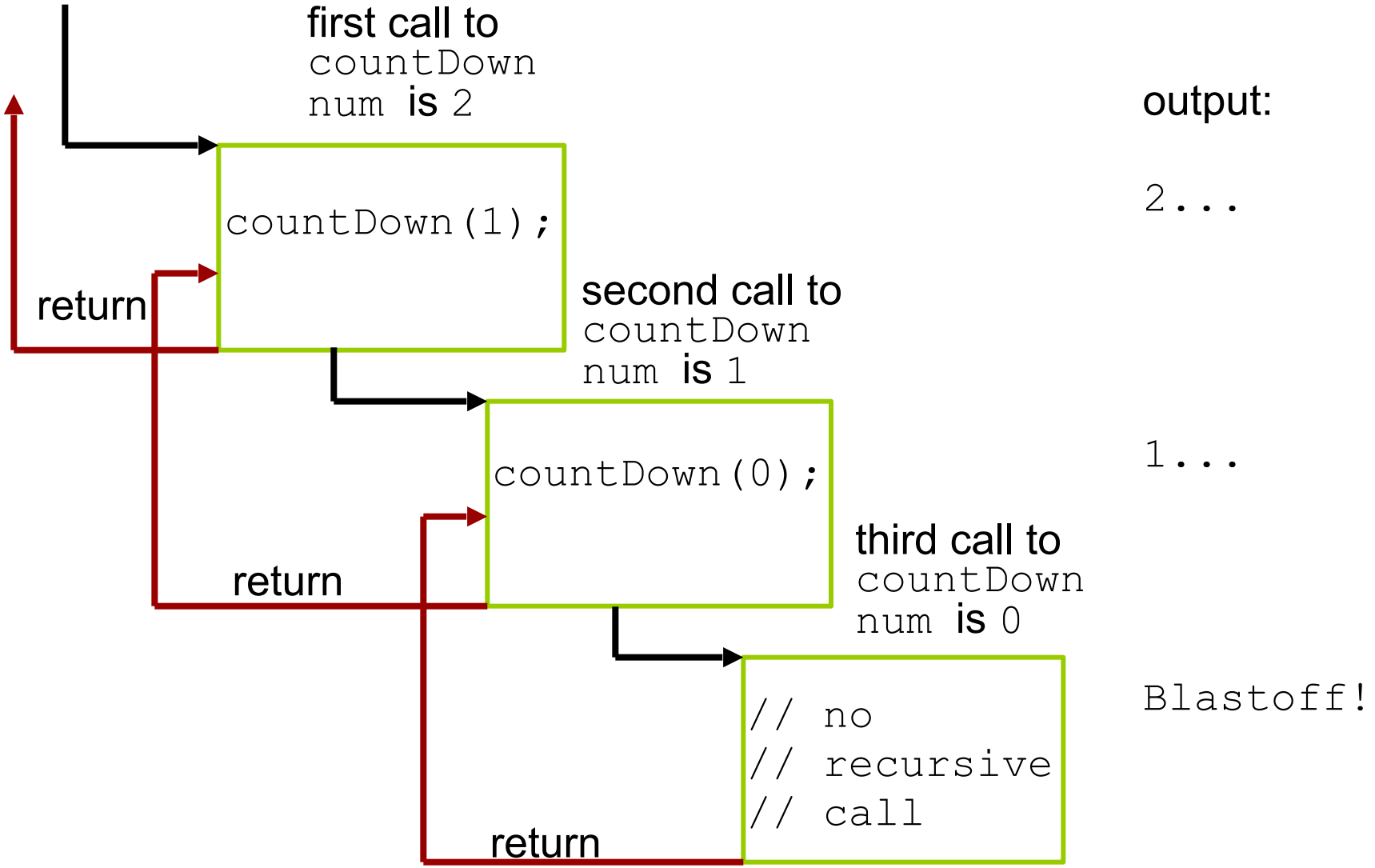
```
void countDown(int num)
{
   if (num == 0)
       cout << "Blastoff!";
   else
   {
    cout << num << "...\n";

    countDown(num-1);  // note that the value
   }                   // passed to recursive
}                      // calls decreases by
                       // one for each call
```

What Happens When Called?

- Each time a recursive function is called, a new copy of the function runs, with new instances of parameters and local variables created
- As each copy finishes executing, it returns to the copy of the function that called it
- When the initial copy finishes executing, it returns to the part of the program that made the initial call to the function

What Happens When Called?



Recursive Function Calls

- What happens if we make changes to the *recursive function* as the following:

```
void countDown(int num)
{
    if (num == 0)
        cout << "Blastoff!";
    else
    {
        // recursive call
        countDown(num-1);

        cout << num << "...\n";
    }
}
```

Types of Recursion

- Direct
  - a function calls itself
- Indirect
  - function A calls function B, and function B calls function A
  - function A calls function B, which calls …, which calls function A

The Recursive Factorial Function

- The factorial function:

  `n! = n*(n-1)*(n-2)*...*3*2*1`, if `n > 0`

  `n! = 1`, if `n = 0`

- Can compute factorial of `n` if the factorial of `(n-1)` is known:

  `n! = n * (n-1)!`

- `n = 0` is the base case

The Recursive Factorial Function

```
int factorial (int num)
{// anchor
 if (num == 0 || num == 1)
    return 1;


 int val = num * factorial(num - 1);
 return val;
}

// let us see the program demo
```

20.3

The Recursive gcd Function

The Recursive gcd Function

- Greatest common divisor (gcd) is the largest factor that two integers have in common
- Computed using Euclid's algorithm:
  `gcd(x, y) = y` if `y` divides `x` evenly
  `gcd(x, y) = gcd(y, x % y)` otherwise
- `gcd(x, y)= y` is the base case

The Recursive gcd Function

```
int gcd(int x, int y)
{
    if (x % y == 0)
       return y;
    else
       return gcd(y, x % y);
}
```

Thank you

Please let me know if you have any further questions!

We are not going to use

Section: 20.7, 20.8, 20.9, 20.10