Functions in C++

Section Signups

- Section signups open tomorrow at 5PM and close Sunday at 5PM.
- Sign up for section at

  http://cs198.stanford.edu/section
- Link available on the CS106B course website.

Getting Started with C++

C++ Functions

- Functions in C++ are similar to methods in Java:
  - Piece of code that performs some task.
  - Can accept parameters.
  - Can return a value.
- Syntax similar to Java:

```
return-type function-name(parameters) {
        /* … function body … */
}
```

Note: no **public** or **private**.

The main Function

- A C++ program begins execution in a function called main with the following signature:

```
int main() {
    /* … code to execute … */
}
```

- By convention, main should return 0 unless the program encounters an error.

A Simple C++ Program

What Went Wrong?

One-Pass Compilation

- Unlike some languages like Java or C#, C++ has a **one-pass compiler**.
- If a function has not yet been declared when you try to use it, you will get a compiler error.
- To fix this, you can do one of two things:
  - Reorder the functions in your source files so that everything is declared before it is used.
  - Use a **function prototype** to tell the compiler to expect a function later on.

Function Prototypes

- A function prototype is a declaration that tells the C++ compiler about an upcoming function.
- Syntax:

    return-type function-name (***parameters***) ;

- A function can be used if the compiler has seen either the function itself or its prototype.

Getting Input from the User

- In C++, we use **`cout`** to display text.
- We can also use **`cin`** to receive input.
- For technical reasons, we've written some functions for you that do input.
  - Take CS106L to see why!
- The library **`"simpio.h"`** contains methods for reading input:

```
int getInteger(string prompt = "");
double getReal(string prompt = "");
string getLine(string prompt = "");
```

Getting Input from the User

- In C++, we use **cout** to display text.
- We can also use **cin** to receive input.
- For technical reasons, we've written some functions for you that do input.
    - Take CS106L to see why!
- The library **"simpio.h"** contains methods for reading input:

```
int getInteger(string prompt = "");
double getReal(string prompt = "");
string getLine(string prompt = "");
```

These functions have default arguments. If you don't specify a prompt, it will use the empty string.

Factorials

- The number **n factorial**, denoted **n!**, is

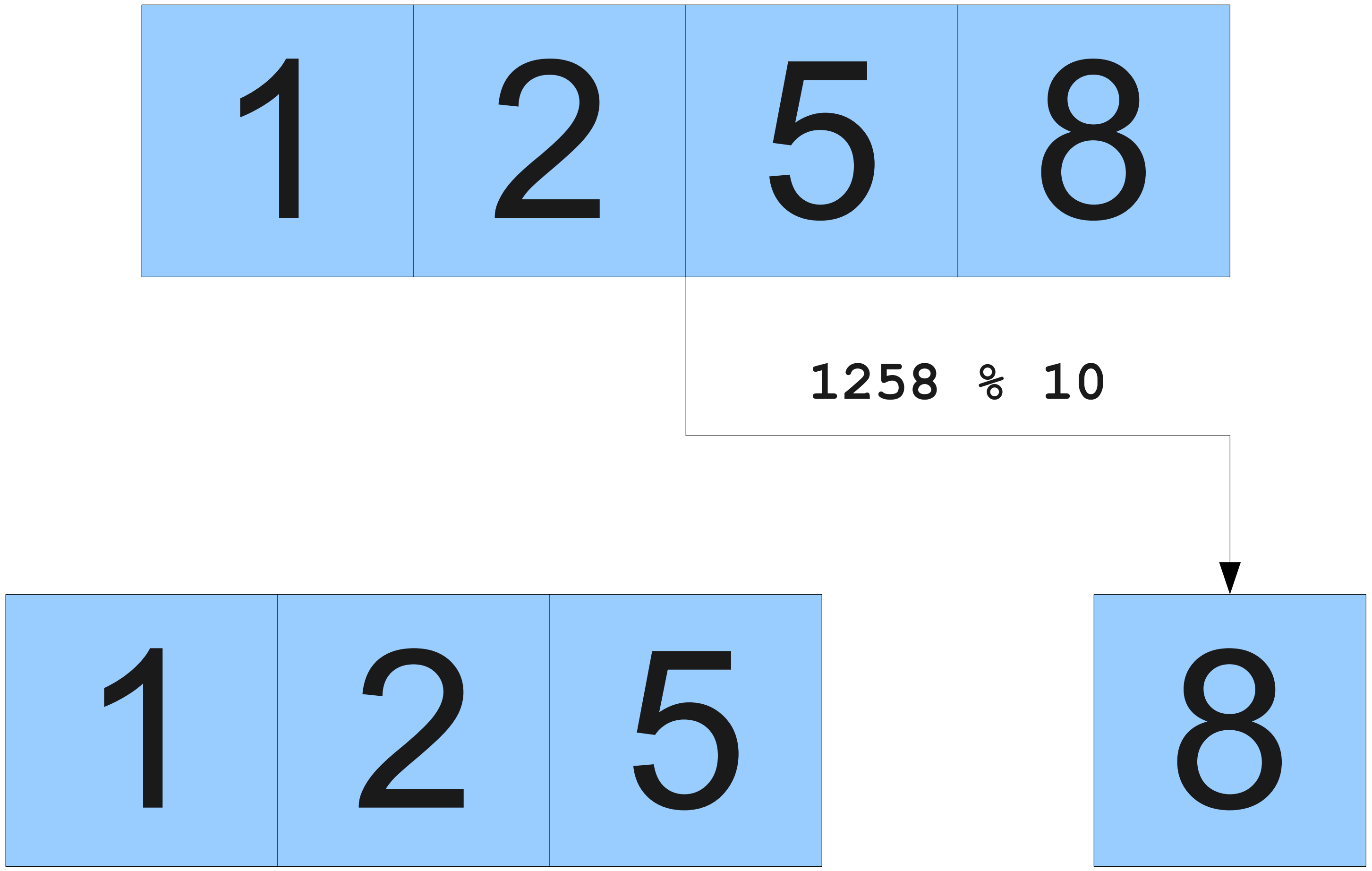
$$n \times (n - 1) \times \ldots \times 3 \times 2 \times 1$$

- For example:
  - $3! = 3 \times 2 \times 1 = 6$.
  - $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$
  - $0! = 1$ (by definition)
- Factorials show up everywhere:
  - Taylor series.
  - Counting ways to shuffle a deck of cards.
  - Determining how quickly computers can sort values (more on that later this quarter).

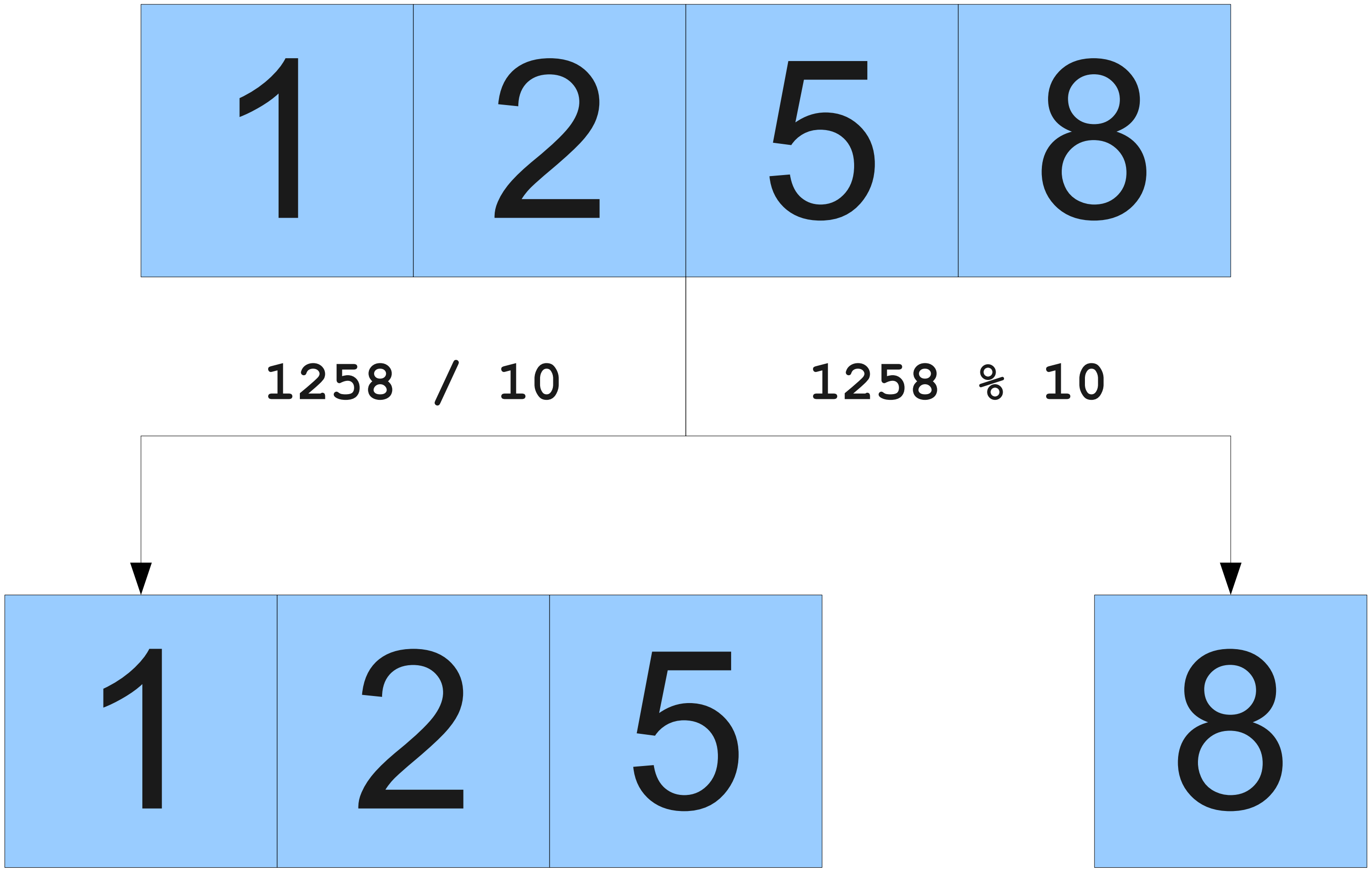
Digital Roots

- The **digital root** of a number can be found as follows:
  - If the number is just one digit, then it's its own digital root.
  - If the number is multiple digits, add up all the digits and repeat.
- For example:
  - 5 has digital root 5.
  - 42 → 4 + 2 = 6, so 42 has digital root 6.
  - 137 → 1 + 3 + 7 = 11, 11 → 1 + 1 = 2, so 137 has digital root 2.

Working One Digit at a Time



Working One Digit at a Time



Thinking Recursively

Factorial Revisited

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

Factorial Revisited

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

Factorial Revisited

$$5! \;=\; 5 \times \underbrace{4 \times 3 \times 2 \times 1}_{4!}$$

Factorial Revisited

$$5! = 5 \times 4!$$

Factorial Revisited

$$5! = 5 \times 4!$$

Factorial Revisited

$$5! = 5 \times 4!$$

$$4! = 4 \times 3 \times 2 \times 1$$

Factorial Revisited

$$5! = 5 \times 4!$$

$$4! = 4 \times 3 \times 2 \times 1$$

Factorial Revisited

$$5! = 5 \times 4!$$

$$4! = 4 \times \underbrace{3 \times 2 \times 1}_{3!}$$

Factorial Revisited

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

Factorial Revisited

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

Factorial Revisited

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2 \times 1$$

Factorial Revisited

$5! = 5 \times 4!$

$4! = 4 \times 3!$

$3! = 3 \times 2 \times 1$

Factorial Revisited

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times \underbrace{2 \times 1}_{2!}$$

Factorial Revisited

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

Factorial Revisited

$$5! = 5 \times 4!$$

$$4! = 4 \times 3!$$

$$3! = 3 \times 2!$$

Factorial Revisited

$$
\begin{aligned}
5! &= 5 \times 4! \\
4! &= 4 \times 3! \\
3! &= 3 \times 2! \\
2! &= 2 \times 1!
\end{aligned}
$$

Factorial Revisited

$$\begin{aligned}
5! &= 5 \times 4! \\
4! &= 4 \times 3! \\
3! &= 3 \times 2! \\
2! &= 2 \times 1! \\
1! &= 1 \times 0!
\end{aligned}$$

Factorial Revisited

$$
\begin{aligned}
5! &= 5 \times 4! \\
4! &= 4 \times 3! \\
3! &= 3 \times 2! \\
2! &= 2 \times 1! \\
1! &= 1 \times 0! \\
0! &= 1
\end{aligned}
$$

Another View of Factorials

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n - 1)! & \text{otherwise} \end{cases}$$

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Another View of Factorials

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n - 1)! & \text{otherwise} \end{cases}$$

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

```
int main() {
    int n = factorial(5);
    cout << "5! = " << n << endl;
}
```

Recursion in Action

```
int main() {
    int n = factorial(5);
    cout << "5! = " << n << endl;
}
```

Recursion in Action

```
int main() {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

int n 5

Recursion in Action

```
int main() {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

int n 5

Recursion in Action

```
int main() {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

int n 5

Recursion in Action

```
int main() {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

int n 5

Recursion in Action

```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

int n: 4

Recursion in Action

```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

```
int n   4
```

Recursion in Action

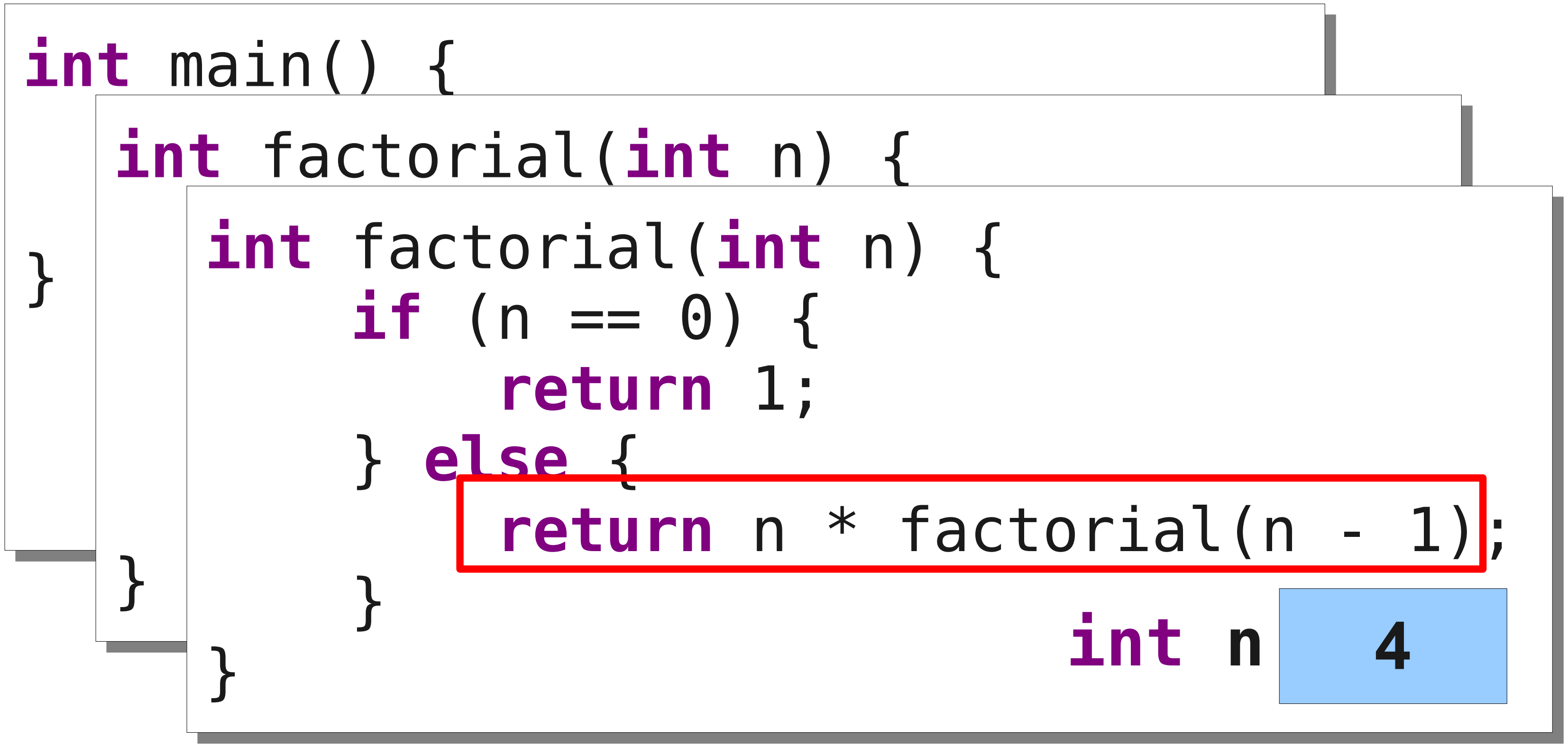
```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

```
int n  4
```

Recursion in Action



```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

```
int main() {

}
```

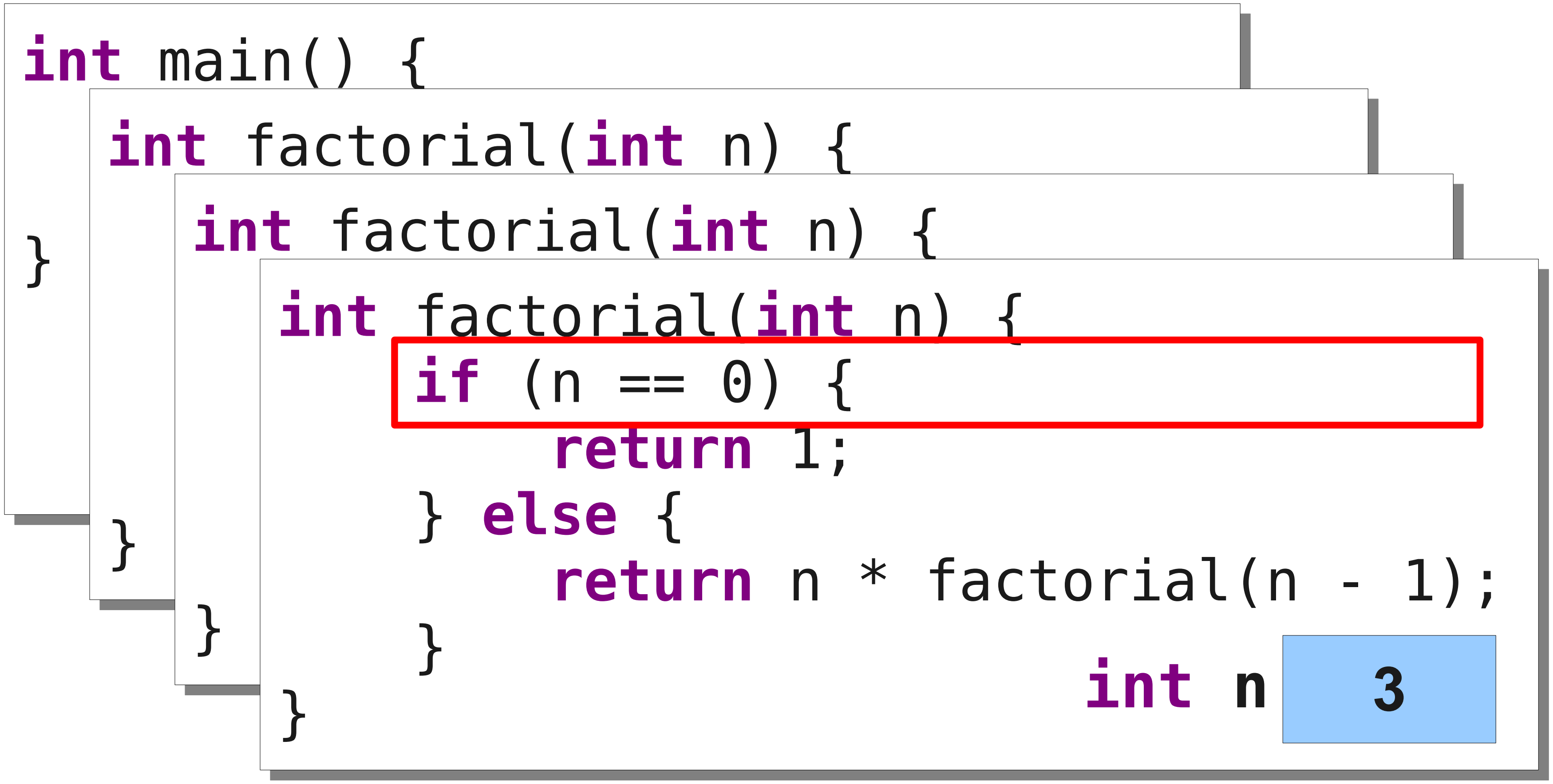
```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

int n 3

Recursion in Action



```
int main() {
}
```

```
    int factorial(int n) {
    }
```

```
        int factorial(int n) {
        }
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

```
int main() {

}
```

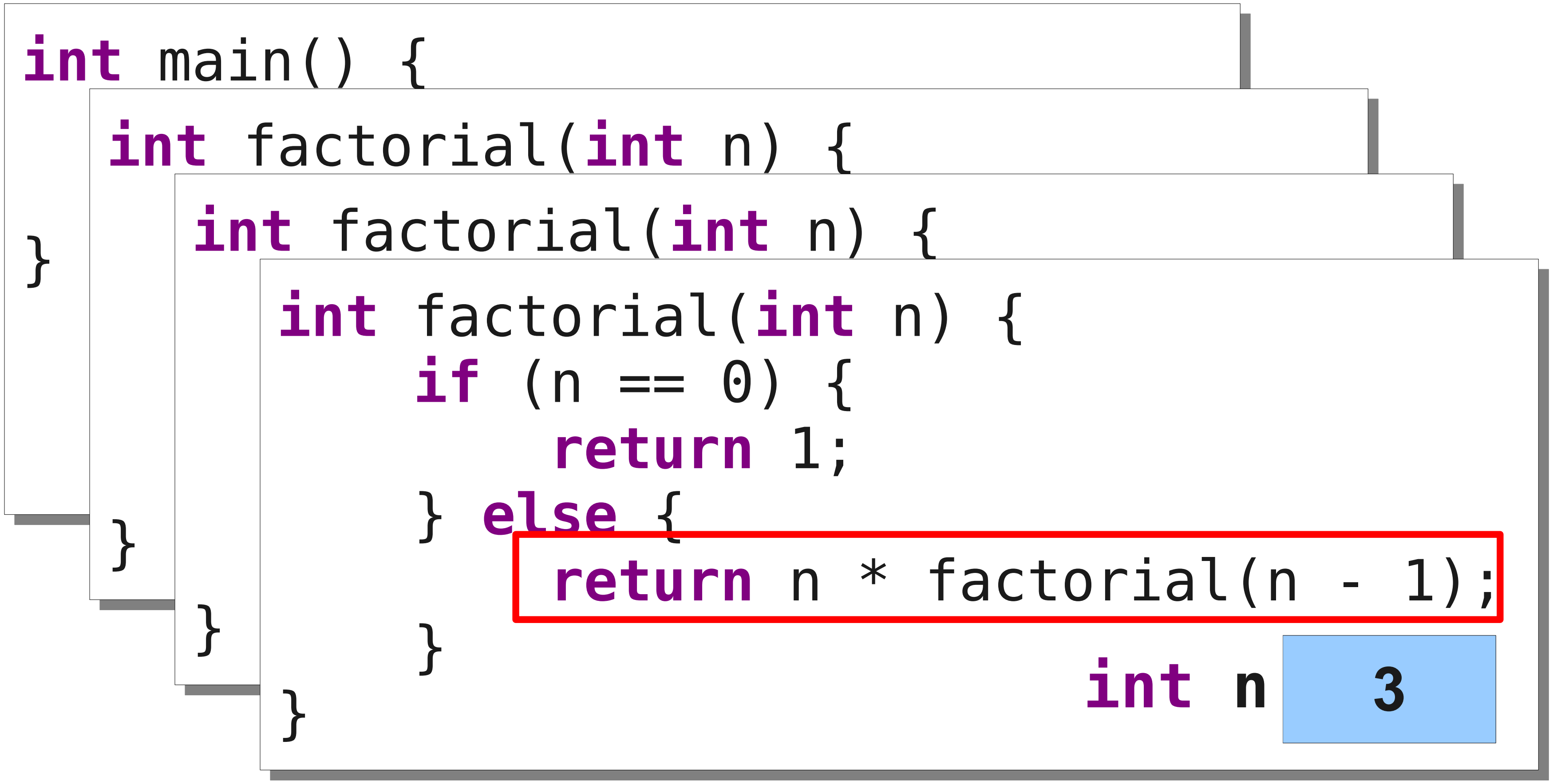
```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

int n 3

Recursion in Action



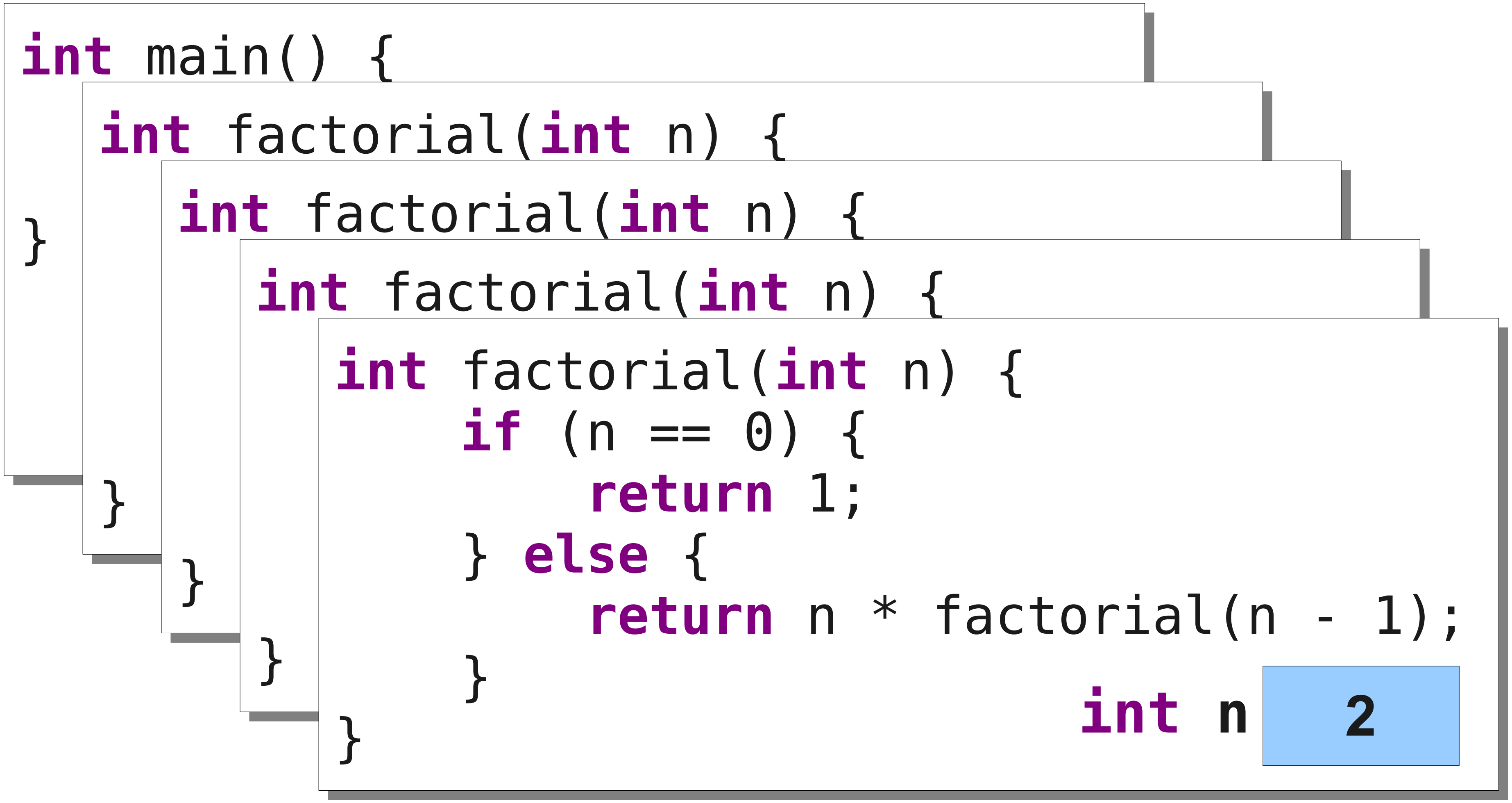
```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action



```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

```
int main() {

}
```

```
int factorial(int n) {

}
```

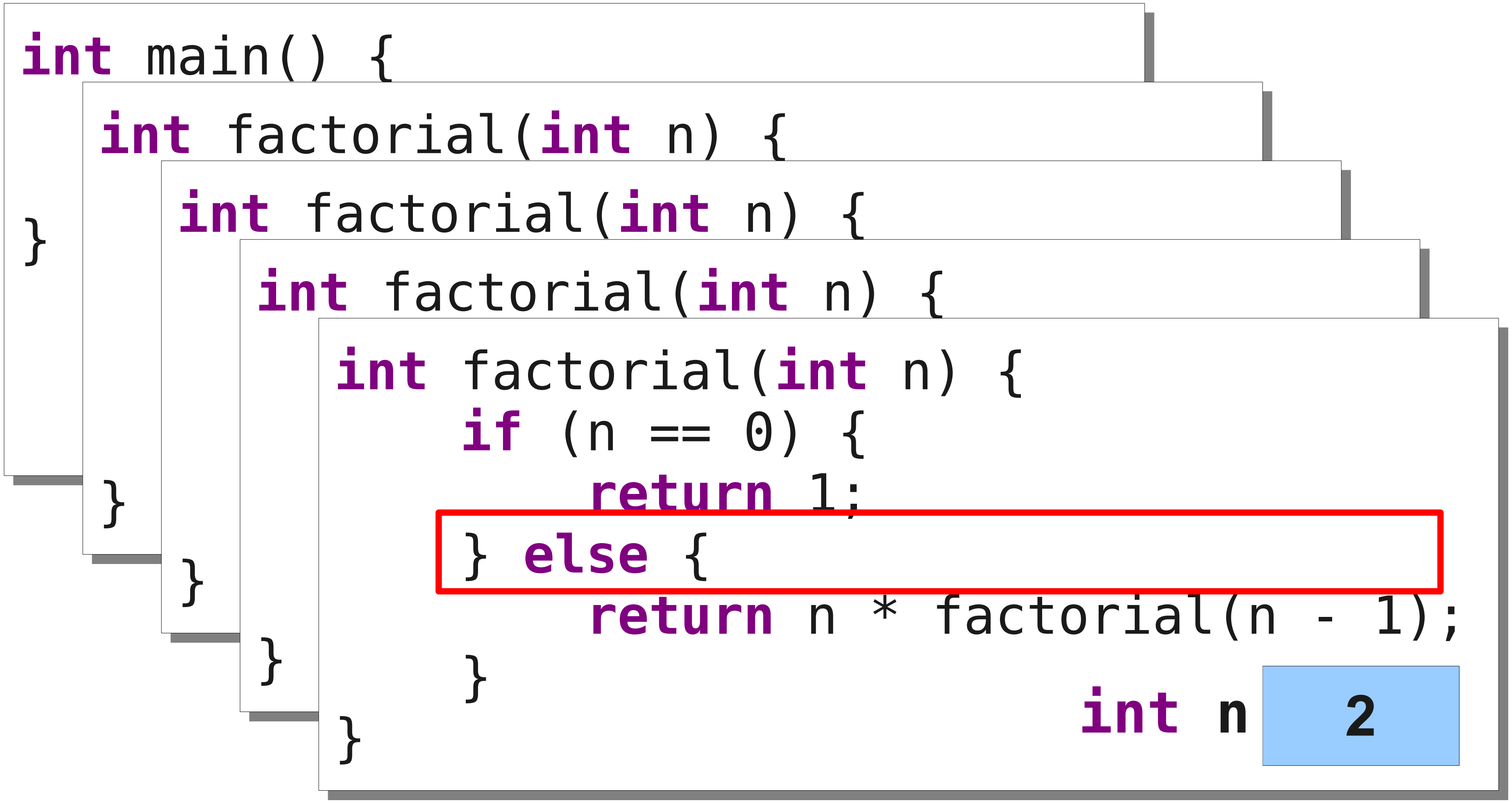
```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

int n | 2

Recursion in Action



```
int main() {

}
```

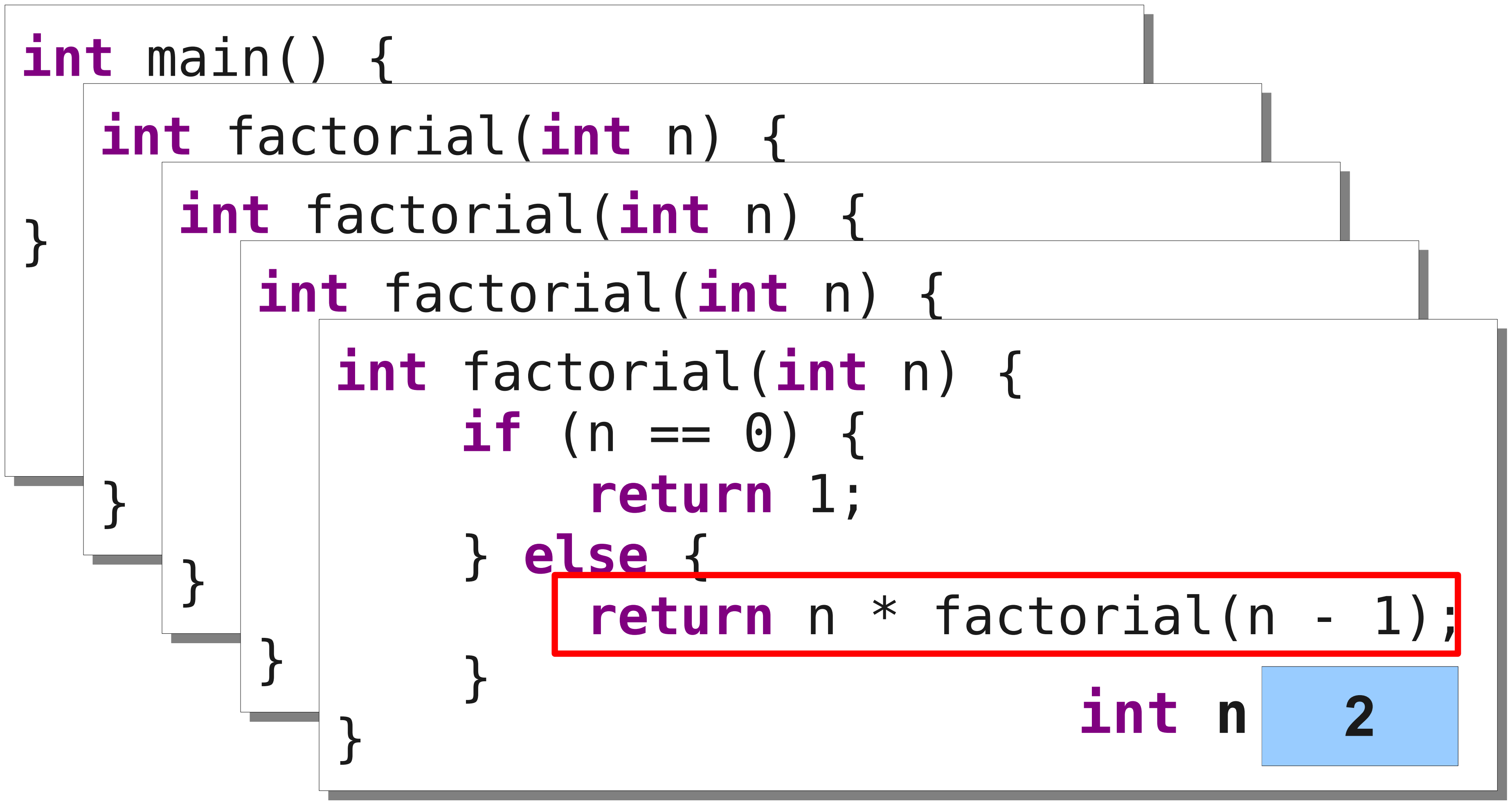
```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action



```
int main() {
}
```

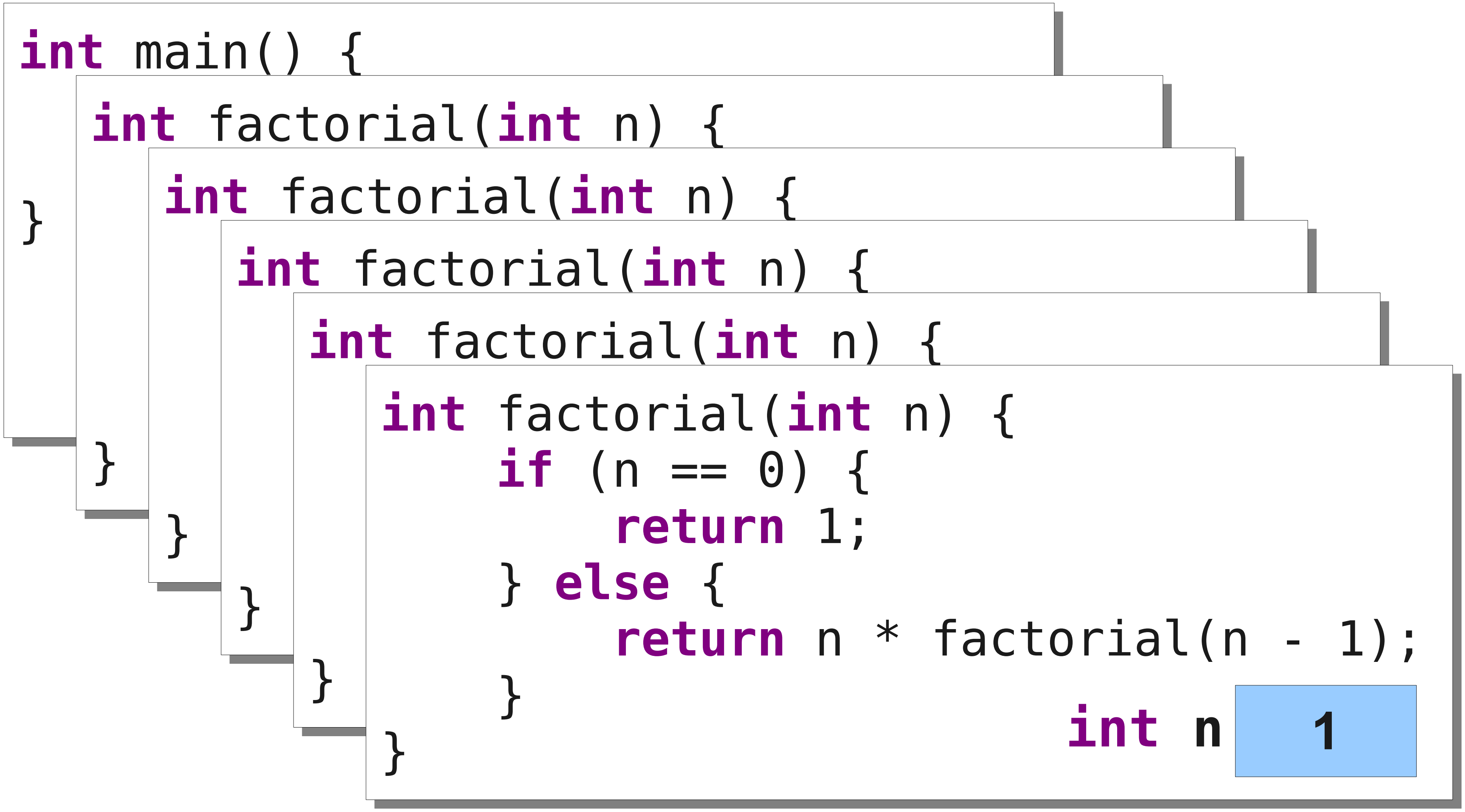
```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action



```
int main() {

}
```

```
int factorial(int n) {

}
```

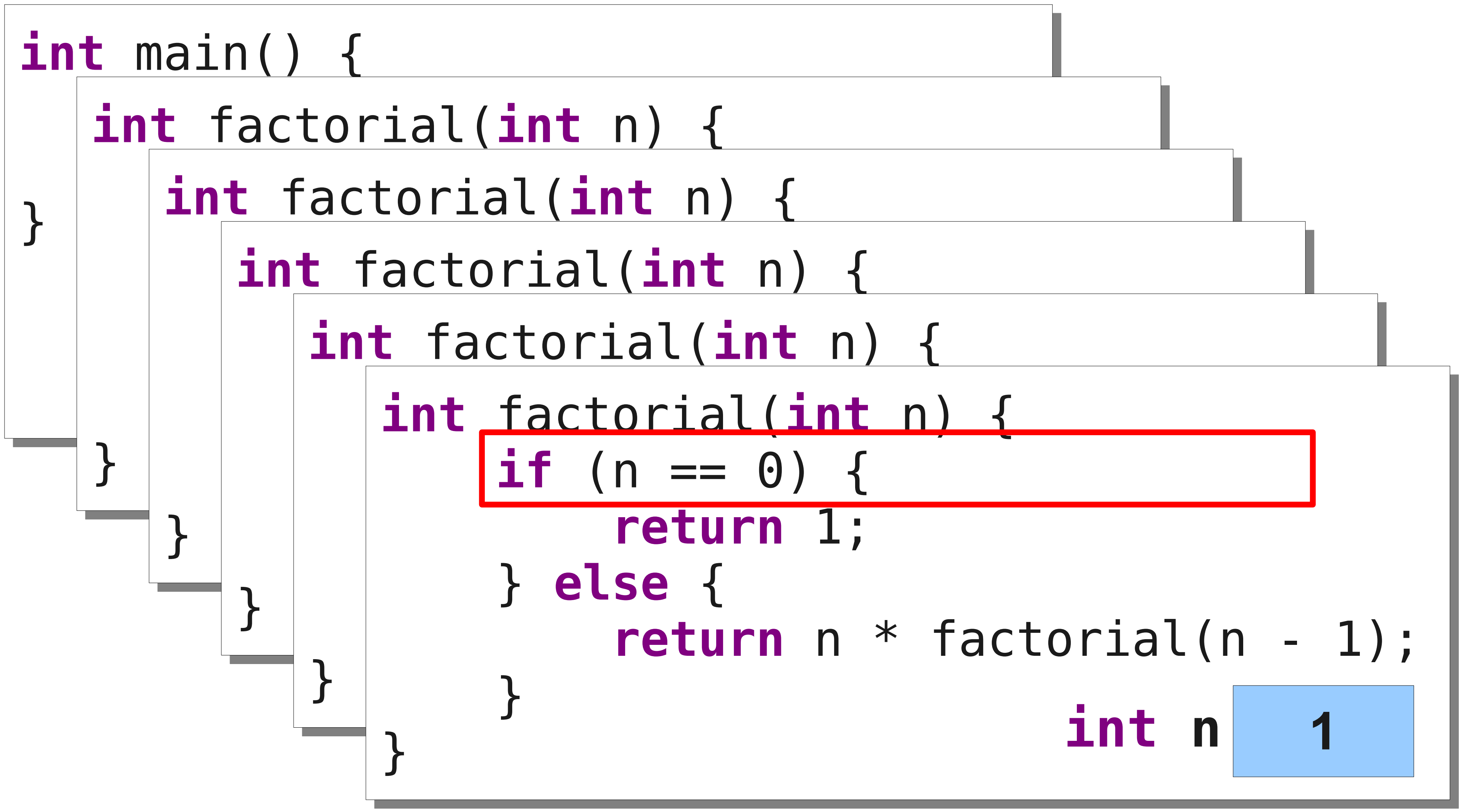
```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action



```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

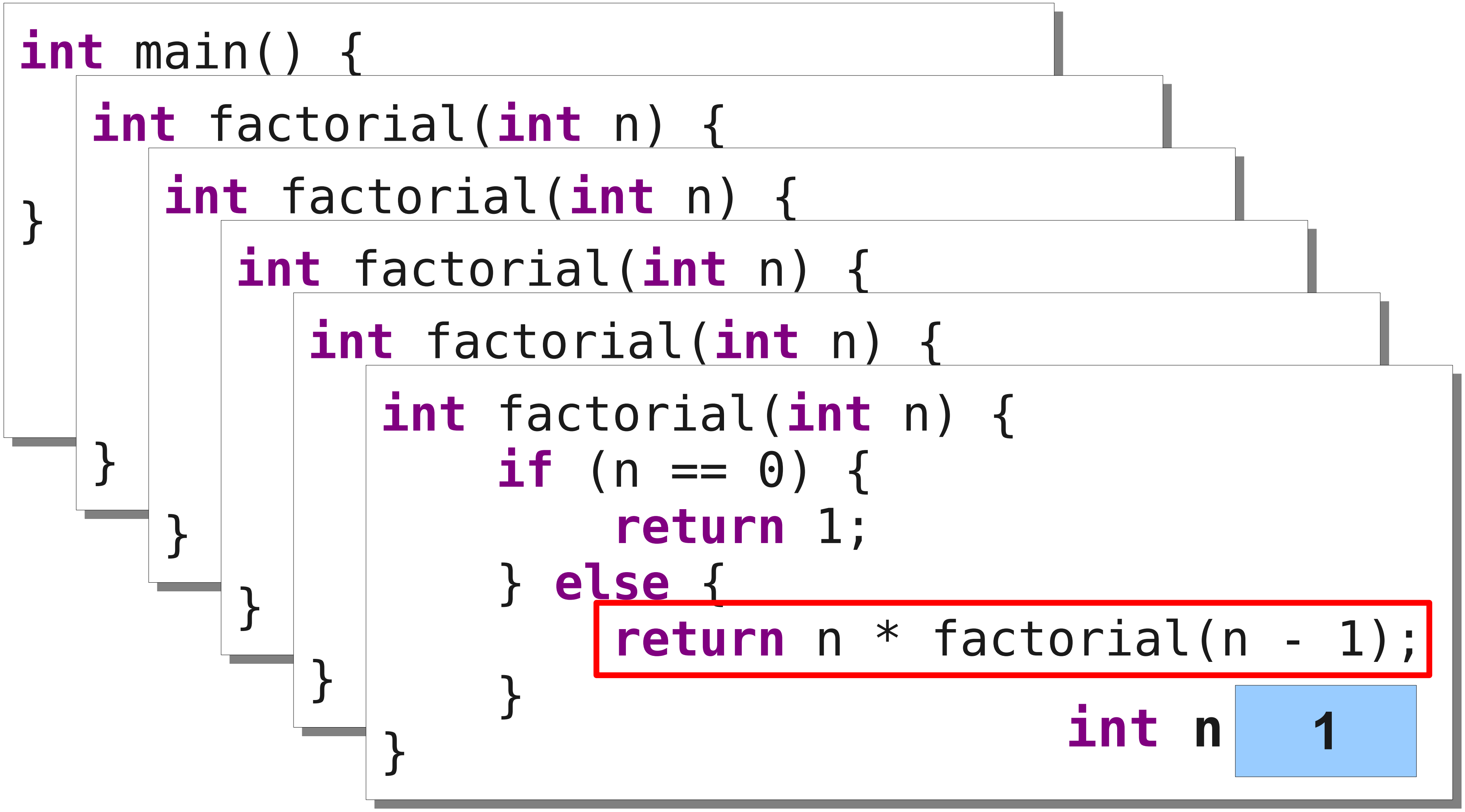
```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

int n 1

Recursion in Action



```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

int n	**0**

Recursion in Action

```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

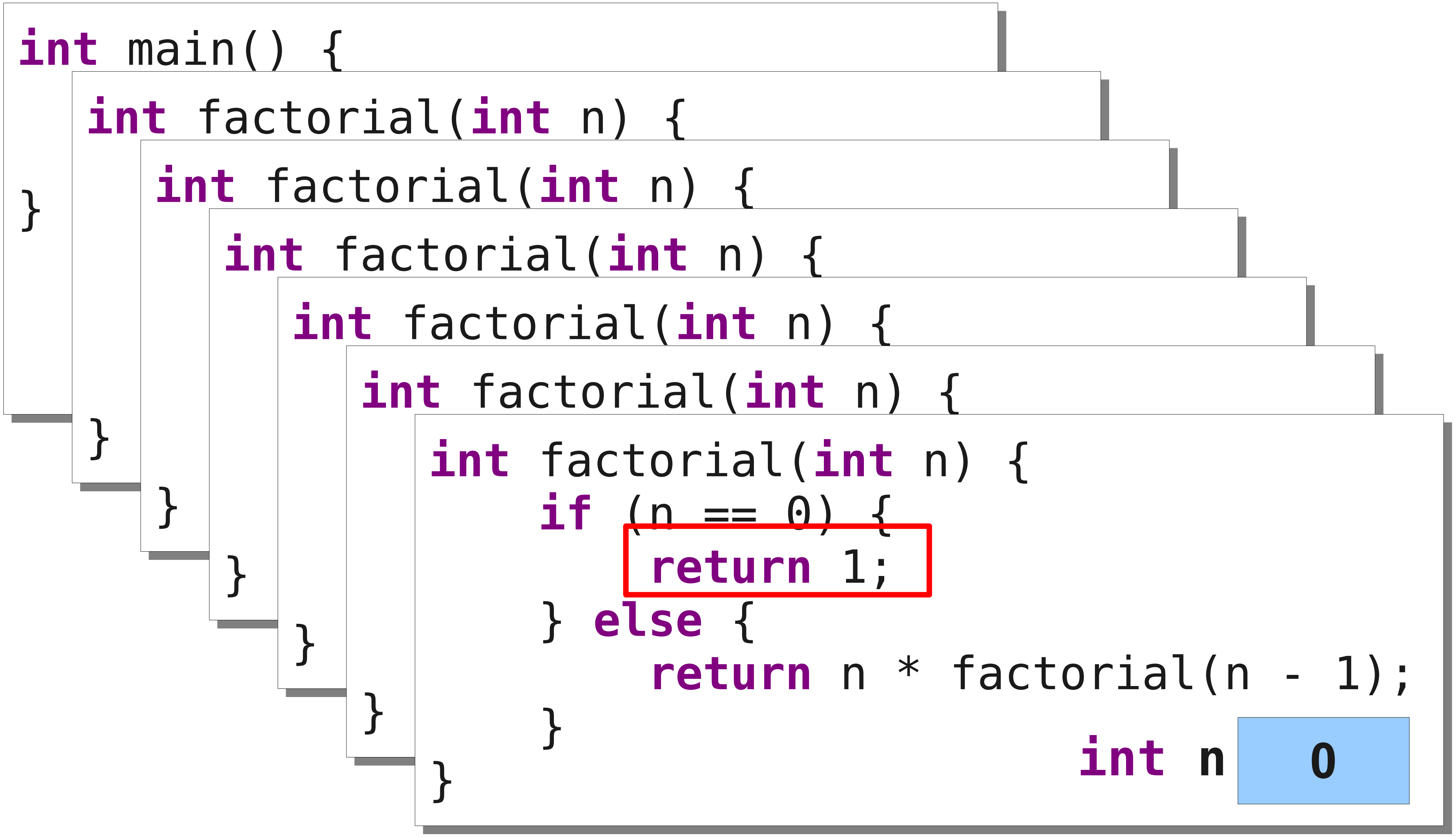
```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
                          int n   0
}
```

Recursion in Action



```
int main() {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

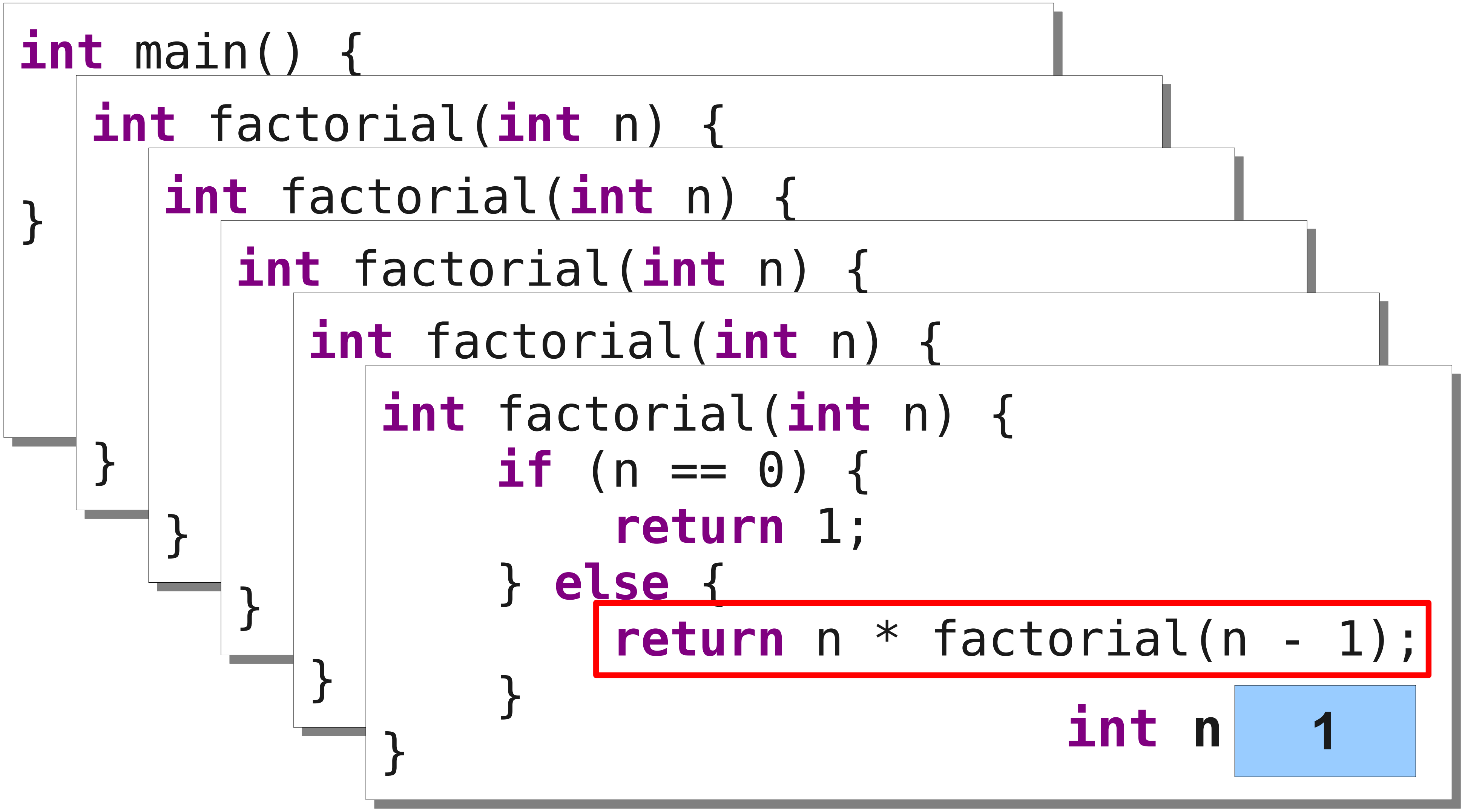
```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action



```
int main() {

}
```

```
int factorial(int n) {

}
```

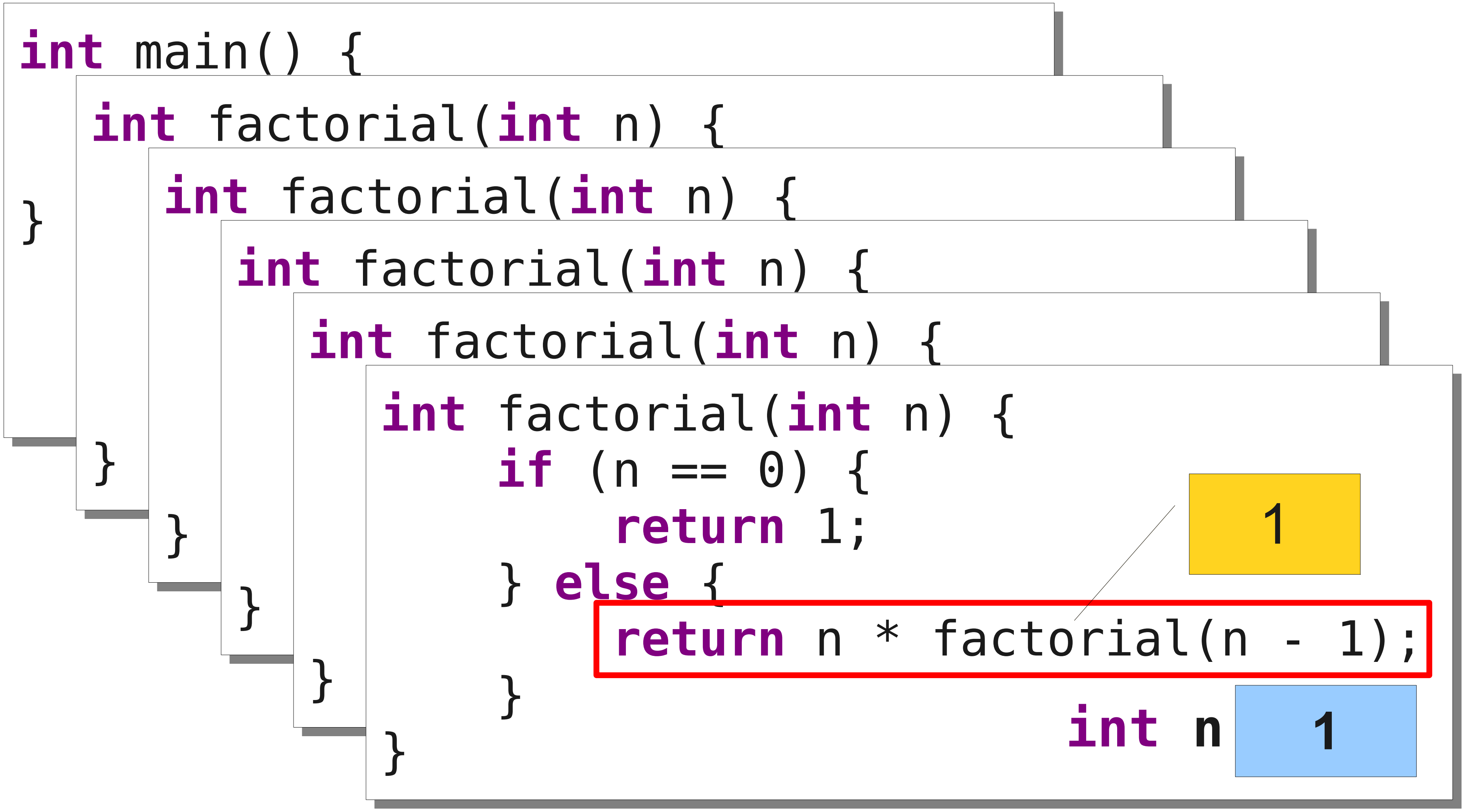
```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

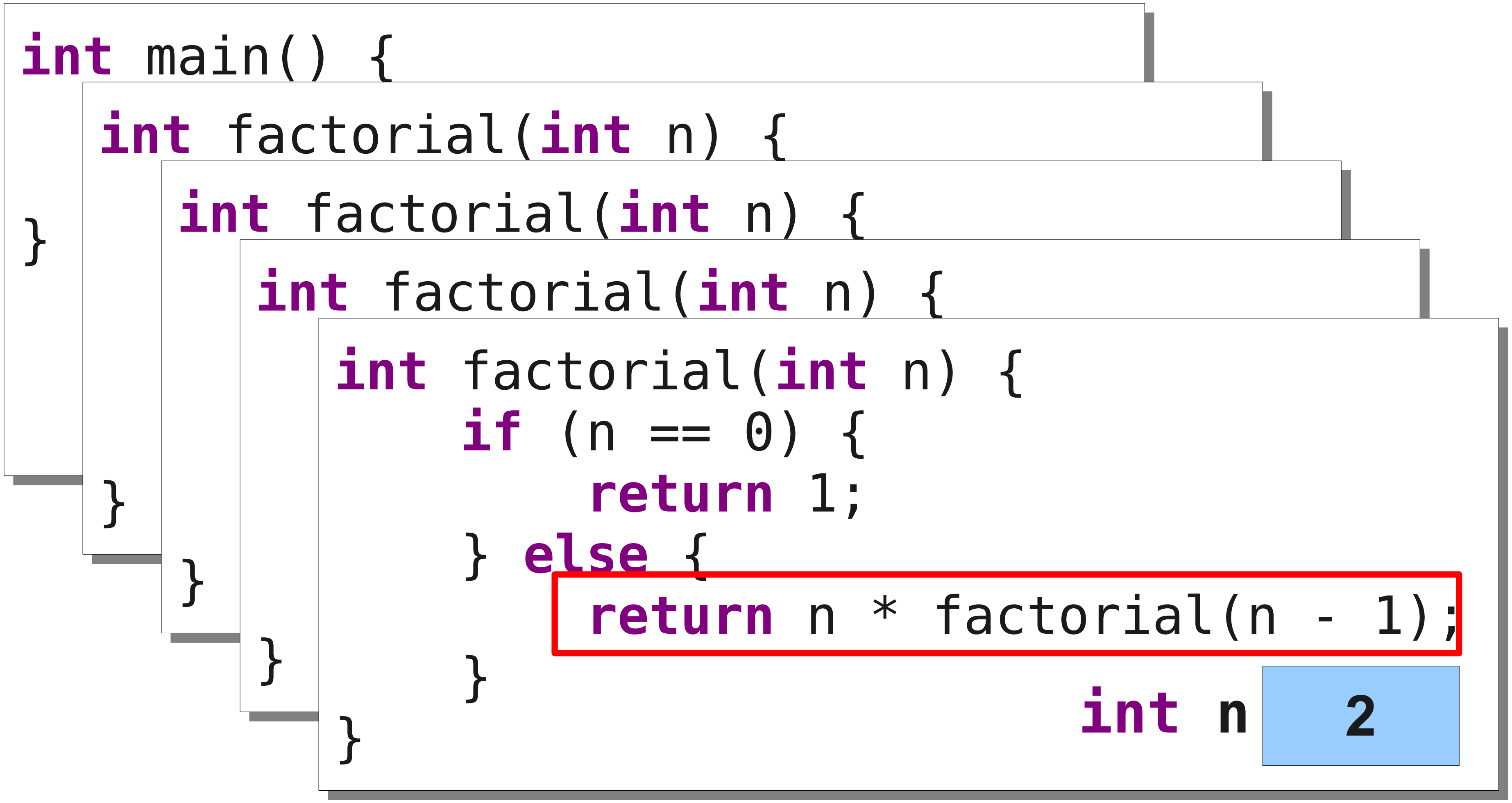


```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action



```
int main() {
}
```

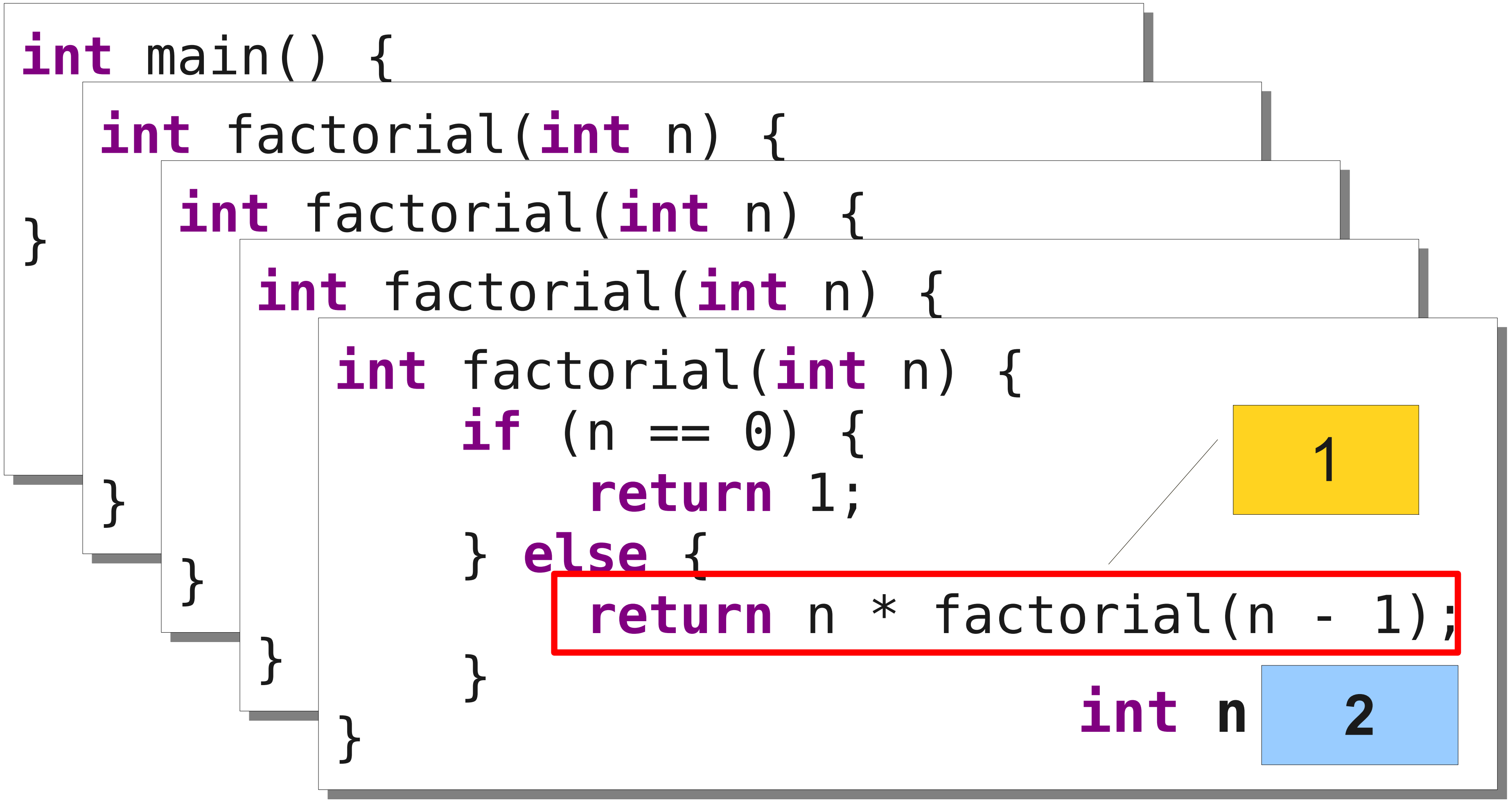
```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

```
int factorial(int n) {
}
```

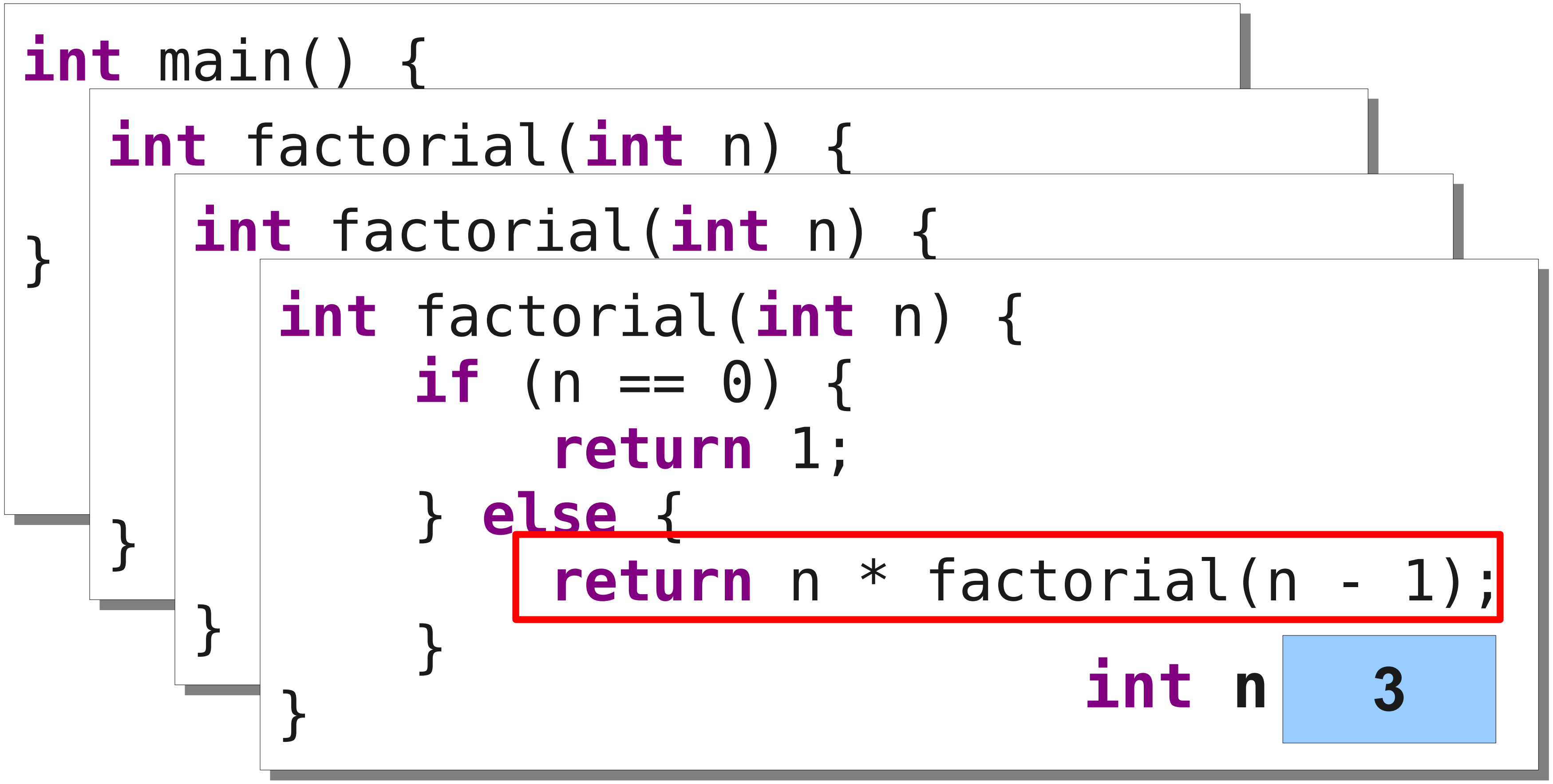
```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action



```
int main() {
    int factorial(int n) {
        int factorial(int n) {
            int factorial(int n) {
                int factorial(int n) {
                    if (n == 0) {
                        return 1;
                    } else {
                        return n * factorial(n - 1);
                    }
                }
            }
        }
    }
}
```

Recursion in Action



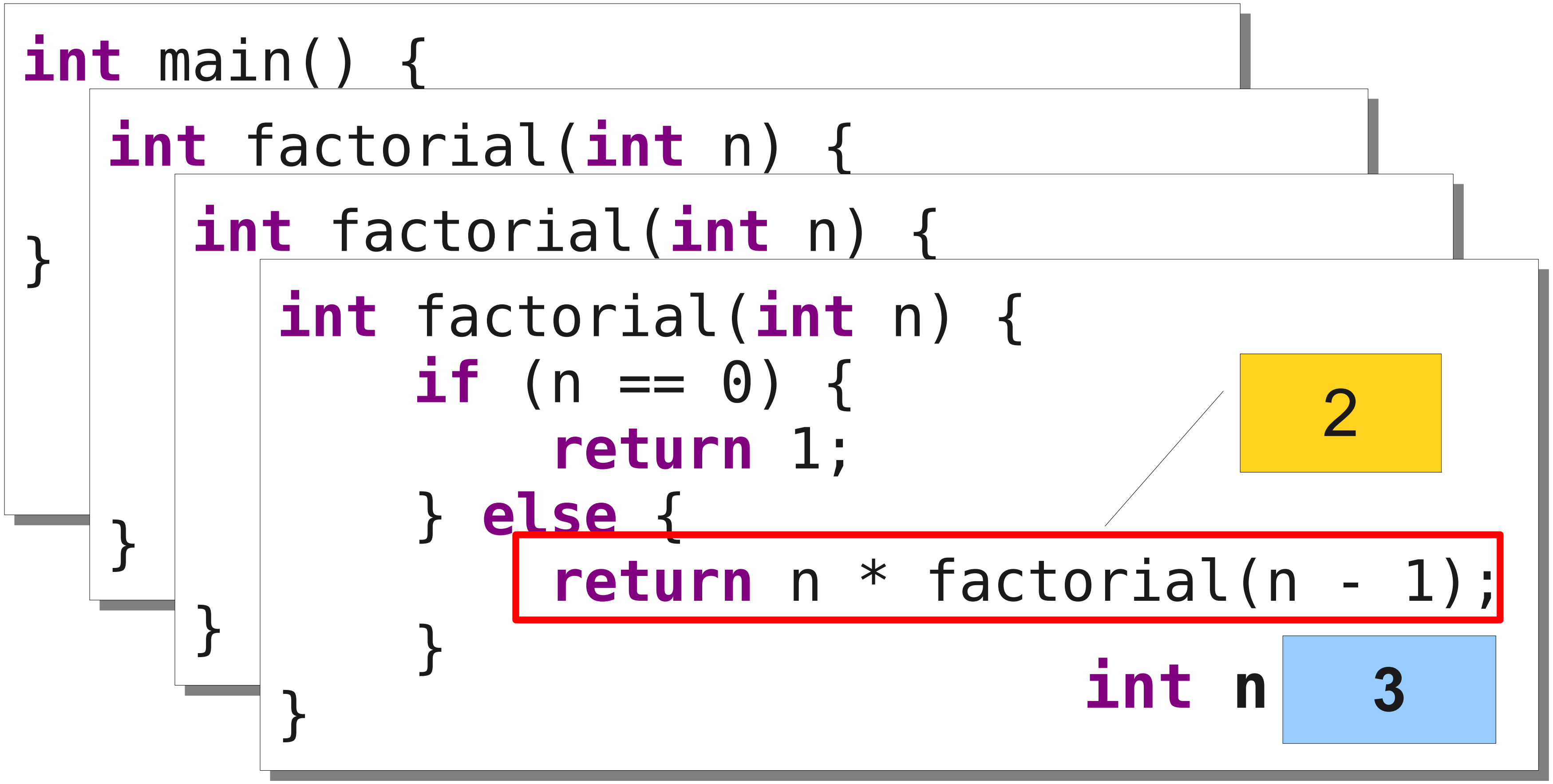
```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action



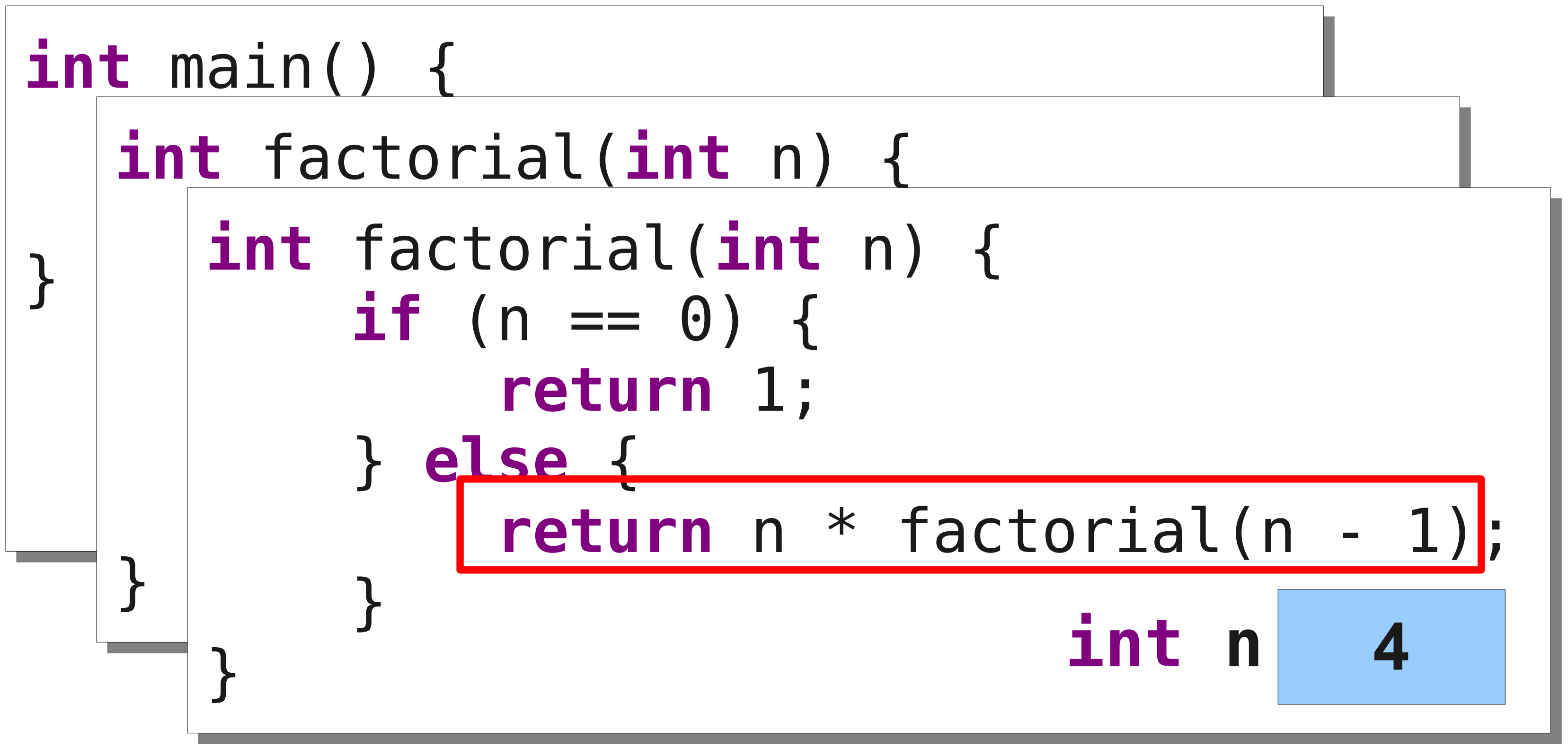
```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

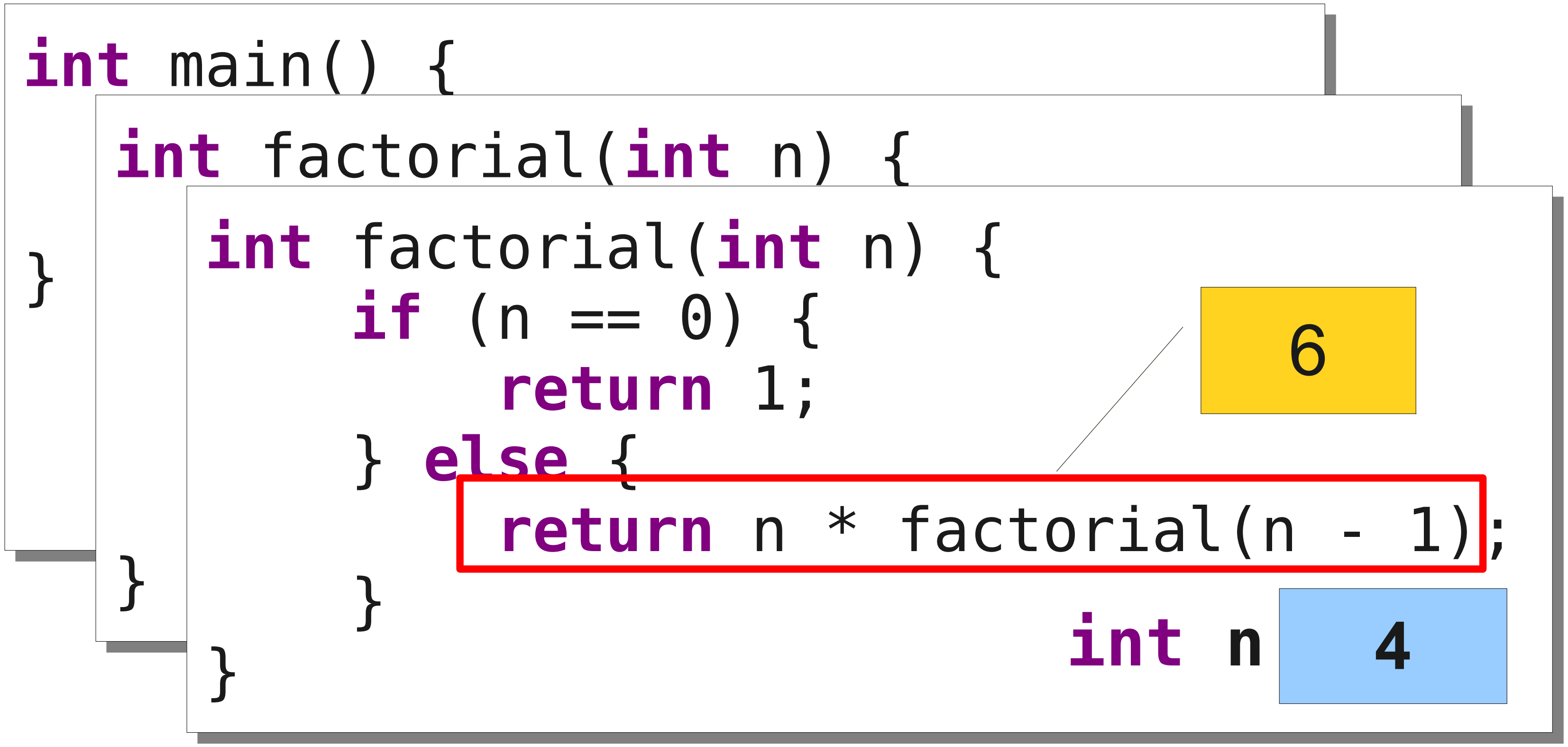


```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

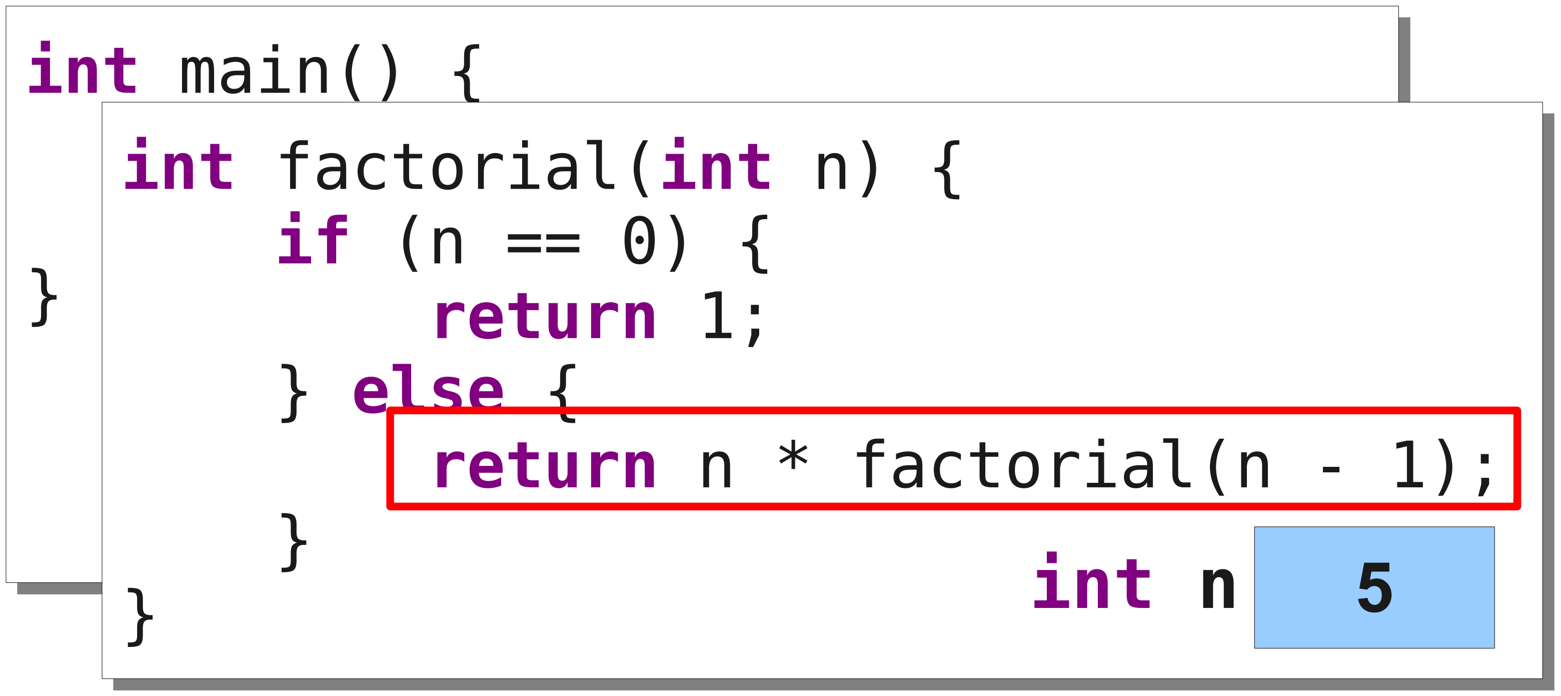


```
int main() {

}
```

```
int factorial(int n) {

}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

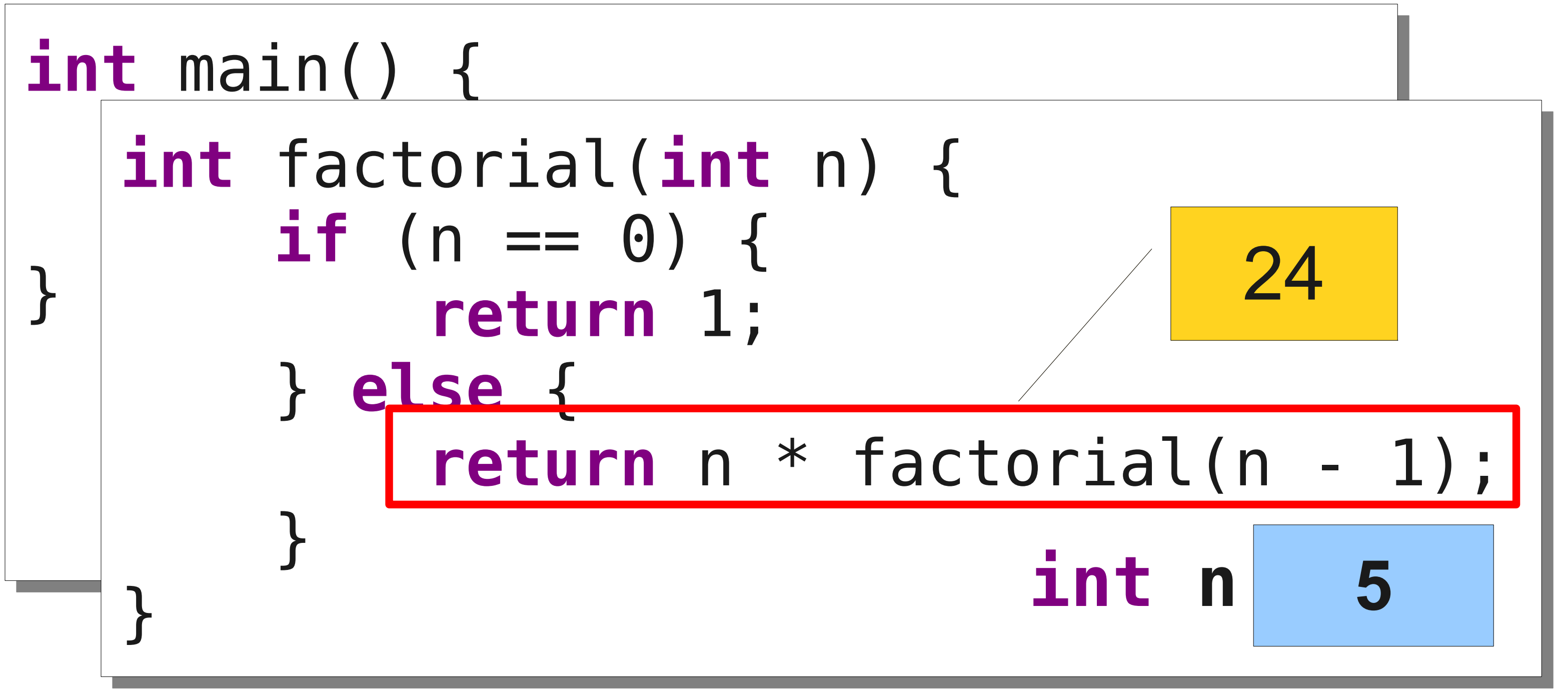
Recursion in Action



```
int main() {
}
```

```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action



```
int main() {

}
```

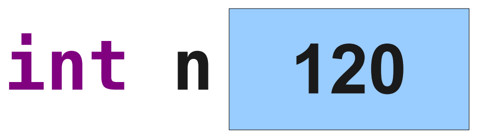
```
int factorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Recursion in Action

```
int main() {
    int n = factorial(5);
    cout << "5! = " << n << endl;
}
```

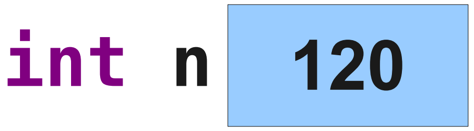
Recursion in Action

```cpp
int main() {
    int n = factorial(5);
    cout << "5! = " << n << endl;
}
```



Recursion in Action

```
int main() {
    int n = factorial(5);
    cout << "5! = " << n << endl;
}
```



Thinking Recursively

- Solving a problem with recursion requires two steps.
- First, determine how to solve the problem for simple cases.
  - This is called the **base case**.
- Second, determine how to break down larger cases into smaller instances.
  - This is called the **recursive decomposition**.

Thinking Recursively

```
if (problem is sufficiently simple) {
    Directly solve the problem.
    Return the solution.
} else {
    Split the problem up into one or more smaller
        problems with the same structure as the original.
    Solve each of those smaller problems.
    Combine the results to get the overall solution.
    Return the overall solution.
}
```

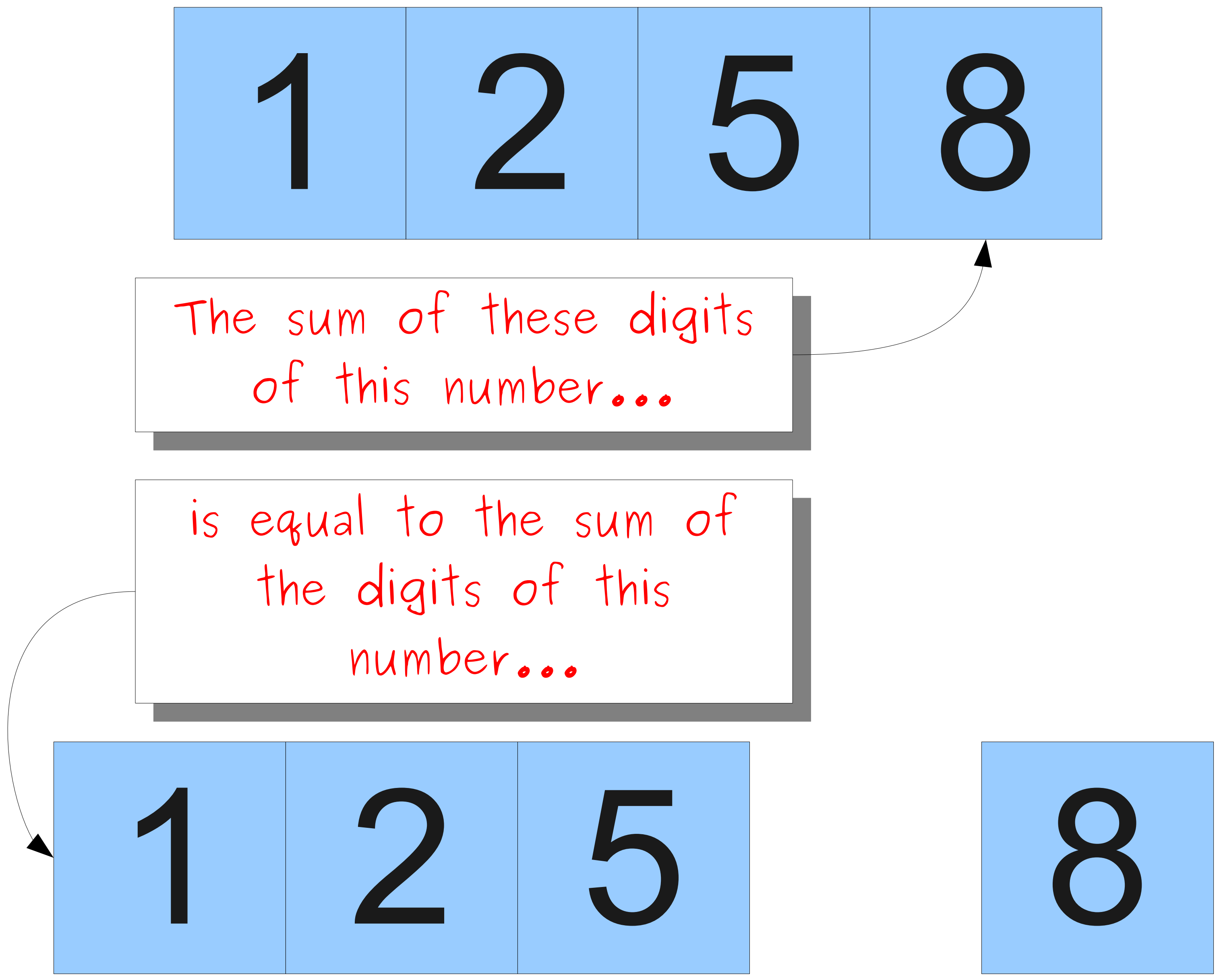
Summing Up Digits

- One way to compute the sum of the digits of a number is shown here:

```
int sumOfDigits(int n) {
    int result = 0;
    while (n != 0) {
        result += n % 10;
        n /= 10;
    }
    return result;
}
```

- How would we rewrite this function recursively?

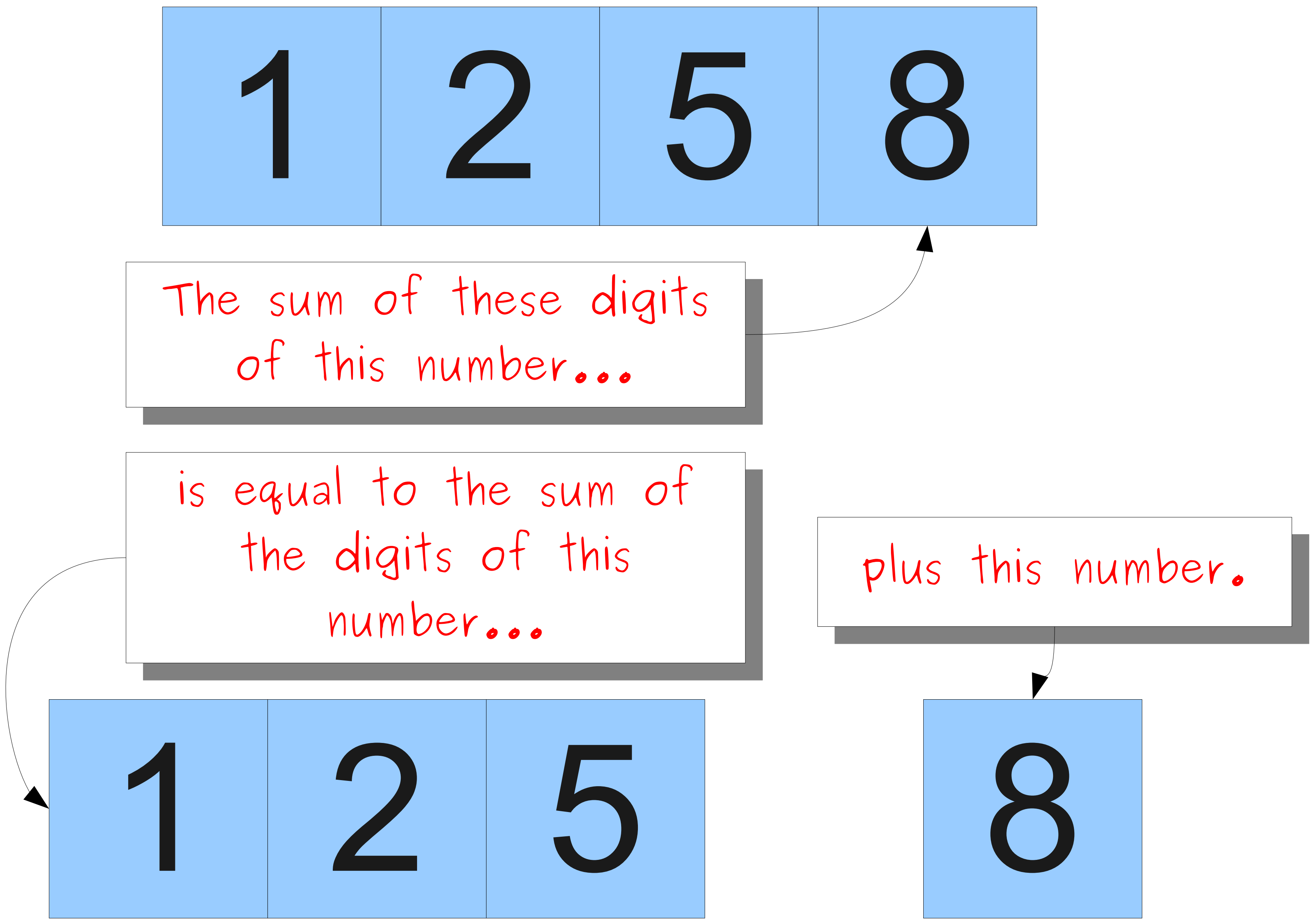
Summing Up Digits



The sum of these digits of this number...

is equal to the sum of the digits of this number...

Summing Up Digits


1 2 5 8
The sum of these digits of this number...
is equal to the sum of the digits of this number...
plus this number.
1 2 5
8

Summing Up Digits

- A recursive implementation of `sumOfDigits` is shown here:

```
int sumOfDigits(int n) {
    if (n < 10) {
        return n;
    } else {
        return (n % 10) + sumOfDigits(n / 10);
    }
}
```

- Notice the structure:
  - If the problem is sufficiently simple, solve it directly.
  - Otherwise, reduce it to a smaller instance and solve that one.

Computing Digital Roots

- One way of computing a digital root is shown here:

```
int digitalRoot(int n) {
    while (n >= 10) {
        n = sumOfDigits(n);
    }
    return n;
}
```

- How might we rewrite this function recursively?

Digital Roots

Digital Roots

The digital root of 9 2 5 8

Digital Roots

The digital root of 9 2 5 8 is the same as

Digital Roots

The digital root of 9 2 5 8 is the same as

The digital root of 9+2+5+8

Digital Roots

The digital root of 9 2 5 8 is the same as

The digital root of 2 4

Digital Roots

The digital root of 9 2 5 8 is the same as

The digital root of 2 4 which is the same as

Digital Roots

The digital root of 9 2 5 8 is the same as

The digital root of 2 4 which is the same as

The digital root of $2 + 4$

Digital Roots

The digital root of 9 2 5 8 is the same as

The digital root of 2 4 which is the same as

The digital root of 6

Computing Digital Roots

- Here is one recursive solution:

```
int digitalRoot(int n) {
    if (n < 10) {
        return n;
    } else {
        return digitalRoot(sumOfDigits(n));
    }
}
```

- Again, notice the structure:
  - Check if the problem is simple enough to solve directly.
  - If not, solve a smaller version of the same problem.

Next Time

- **Strings and Streams**
  - Representing and manipulating text.
  - File I/O in C++.