Functions in C++

Website is Up!!!

Website is Up!!!

CS106B

Programming Abstractions in C++

Welcome to CS106B!

June 24, 2013

Welcome to CS106B! We've got an exciting quarter ahead of us and you're in for a real programming treat. Over the next ten weeks, we'll be exploring the fundamental techniques necessary to reason about, model, and solve big, important problems. It's going to be a lot of fun, and I hope that you're able to join us!

In the meantime, feel free to check out the **course information handout** and **syllabus** to learn more about what this class is all about, the prerequisites, and the course policies. If you have any questions in the meantime, feel free to email me at **adgress@cs.stanford.edu** with questions.

See you soon!

Handouts

00: Course Information
01: Syllabus
02: Course Placement
03L: Running C++ On Linux
03M: Running C++ On Mac
03W: Running C++ On Windows
04: Honor Code
06M: Debugging with Xcode
06W: Debugging with Visual Studio
07: Submitting Assignments

Section Handouts

Assignments

Resources

Course Reader PDF
Tresidder LaIR Office Hours
C and C++ Standard Library Docs
Stanford C++ Library Docs
Good Programming Style 1
Good Programming Style 2
Submitter
Lecture Videos
QuestionHut
Blank Windows Project
Blank Mac Project

Lectures

Today

- Getting Started in C++
- Thinking Recursively
- Style Gameshow
- Parameter Passing and Common Mistakes

Today

- Getting Started in C++
- Thinking Recursively
- Style Gameshow
- Parameter Passing and Common Mistakes

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.

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.

hello-world.cpp (On Board)

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`.

abs.cpp
(On Computer)

What Went Wrong?

One-Pass Compilation

- Unlike some languages like Java or C#, C++ has a **one-pass compiler**.
  - Think of it like a person reading a book from start to finish.
- If a function has not yet been declared when you try to use it, you will get a compiler error.

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.

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).

factorial.cpp
(On Board)

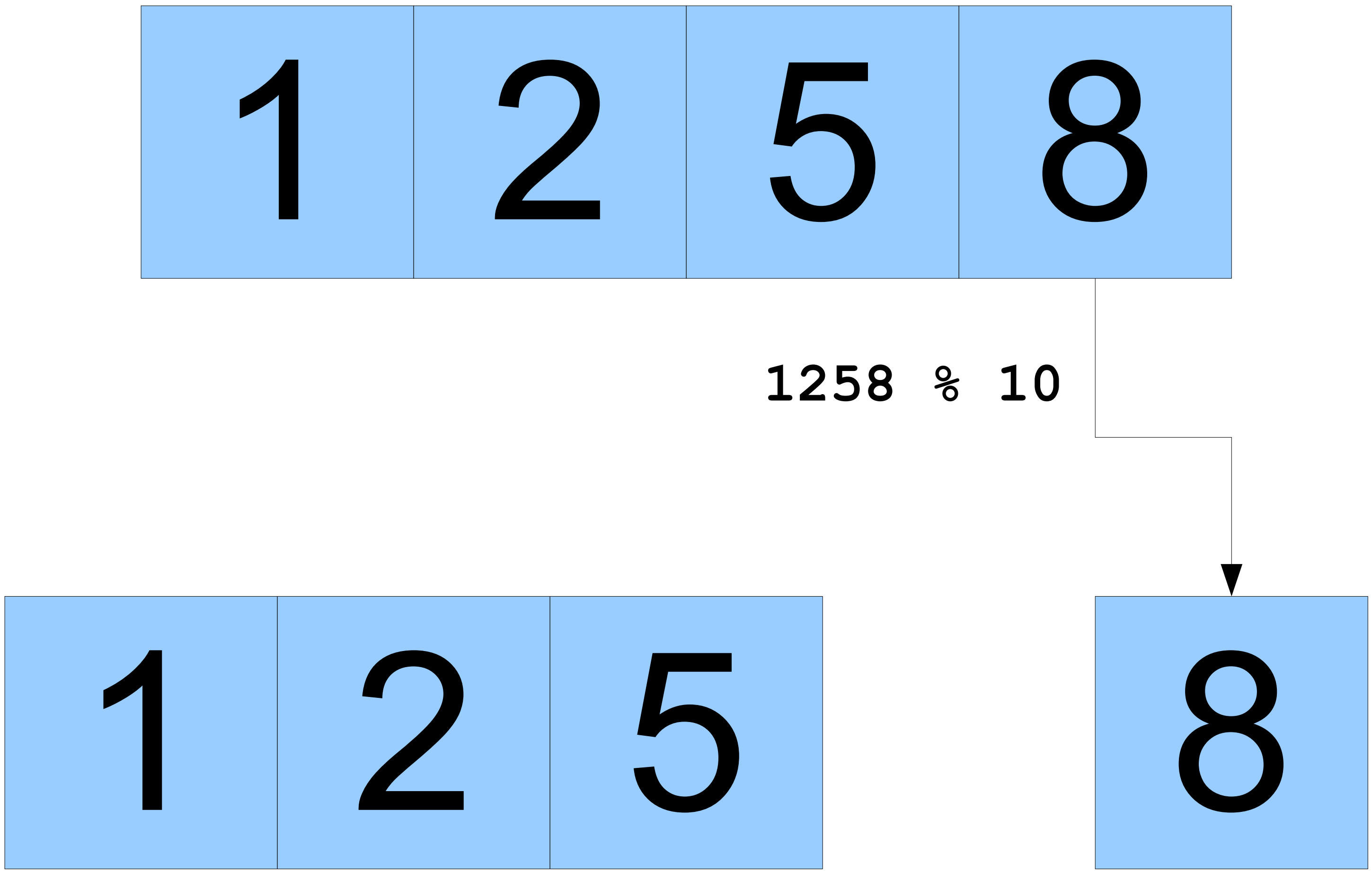
Digital Roots

- 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.
  - $42 \rightarrow 4 + 2 = 6$, so 42 has digital root 6.

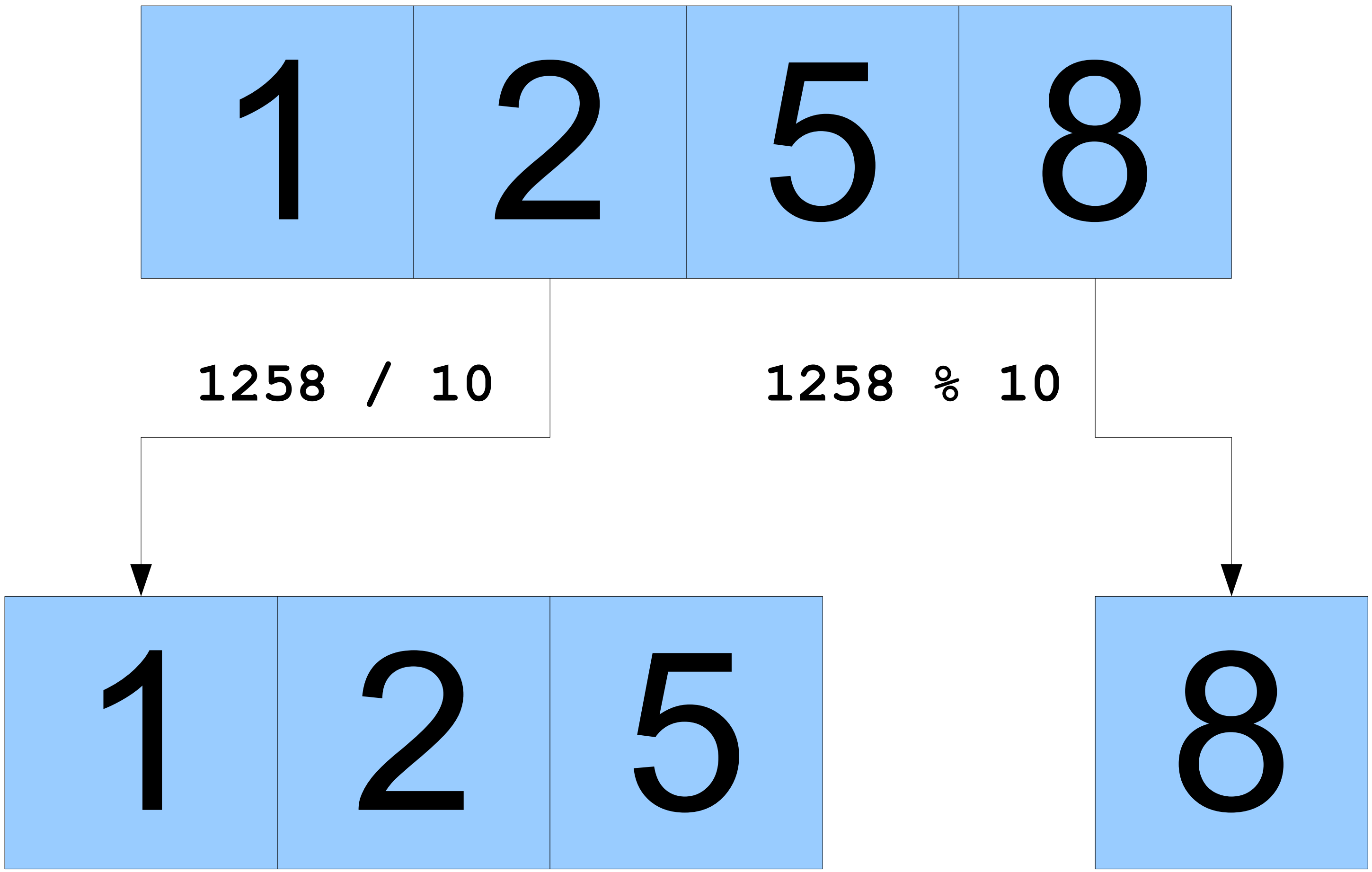
Digital Roots

- 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



digital-root.cpp
(On Board)

Announcements

- Lectures are recorded. Link on website.
- Lecture slides and code are posted on the website.
- Correction: LAIR hours are 7-11pm, Sunday-Wednesday (starting this Wednesday)
- No section this week
- More course readers in bookstore Wednesday ~Noon

More Announcements...

- Five Handouts Today:
  - Downloading XCode/Visual Studio/g++
  - **Honor Code**
  - **Assignment 1: Welcome to C++!**
  - Submitting Assignments
  - Debugging with Visual Studio/Xcode
- Assignment 1 (Welcome to C++!) out later today, due Tuesday, July 2 at 11AM.
  - **Starter files are being updated, email will be sent when ready**

The CS106B Grading Scale

++

+

✓+

✓

✓-

-

--

0

Assignment Grading

- You will receive two scores: a functionality score and a style score.
- The **functionality score** is based on correctness.
  - Do your programs produce the correct output?
  - Do they work on all legal inputs?
- The **style score** is based on how well your program is written.
  - Are your programs well-structured?
  - Do you use variable naming conventions consistently?

Late Days

- Everyone has **four** free “late days” to use as needed.
- A “late day” is an automatic 24 hour extension.
- If you need an extension beyond late days, please talk to Aubrey.
  - We generally only give extra extensions for medical reasons.
- **Max days an assignment can be late is 3.** Past this we won't grade it.

Honor Code

- Unfortunately the Computer Science department has a disproportionately high number of honor code violations.
- The most likely reason for this is that we are very good at detecting honor code violations (we have automated tools that do this for us).

Honor Code

- Handout on the honor code is on the website. Please read it.
- The overwhelming majority of the honor code for CS106B is:
  - Don't look at over students' (past or present) code.
  - Don't show your code to any else in the class.
  - If your code is based off of something you found online or in the course reader, then cite the source.

Today

- Getting Started in C++
- Thinking Recursively
- Style Gameshow
- Parameter Passing and Common Mistakes

A **recursive solution** is a solution that is defined in terms of itself.

Recursion: Fibonacci Numbers

- Fibonacci Numbers
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, …
  - Defined *recursively*:

$$fib(n) = \begin{cases} n & \text{if } n = 0 \text{ or } 1 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

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

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

Factorial Revisited

$$5! = 5 \times 4!$$
$$4! = 4 \times 3!$$
$$3! = 3 \times 2!$$
$$2! = 2 \times 1!$$
$$1! = 1 \times 0!$$

Factorial Revisited

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

factorial.cpp
(On Computer)

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() {

}
```

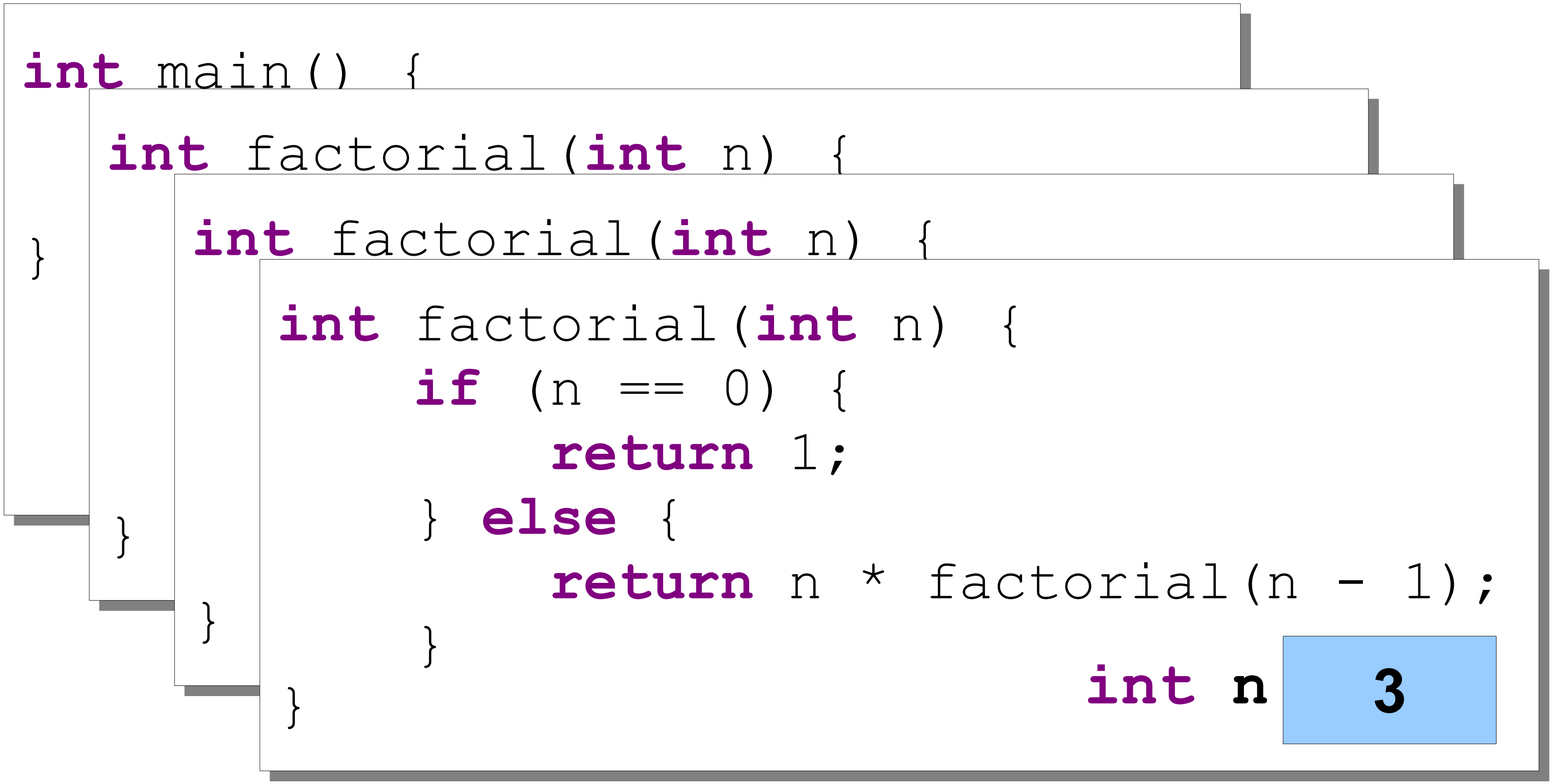
```
int factorial(int n) {

}
```

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

int n 4

Recursion in Action



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);
    }
}
```

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);
    }
}
```

int n 3

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);
    }
}
```

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);
    }
}
```

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);
    }
}
```

int n = 2

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);
                        }
                    }
                }
            }
        }
    }
}
```

int n: 1

Recursion in Action

```
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);
    }
}
```

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) {

}
```

```
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);
    }
}
```

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

int n: 1

Recursion in Action

```
int main() {

}
```

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

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) {
    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);
    }
}
```

1

int n 2

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);
    }
}
```

2

int n 3

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);
    }
}
```

6

int n 4

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);
    }
}
```

24

int n 5

Recursion in Action

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

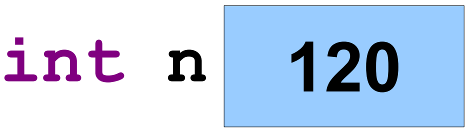
Recursion in Action

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

int n 120

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**.

Another View of Factorials

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

Base Case

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.
}
```

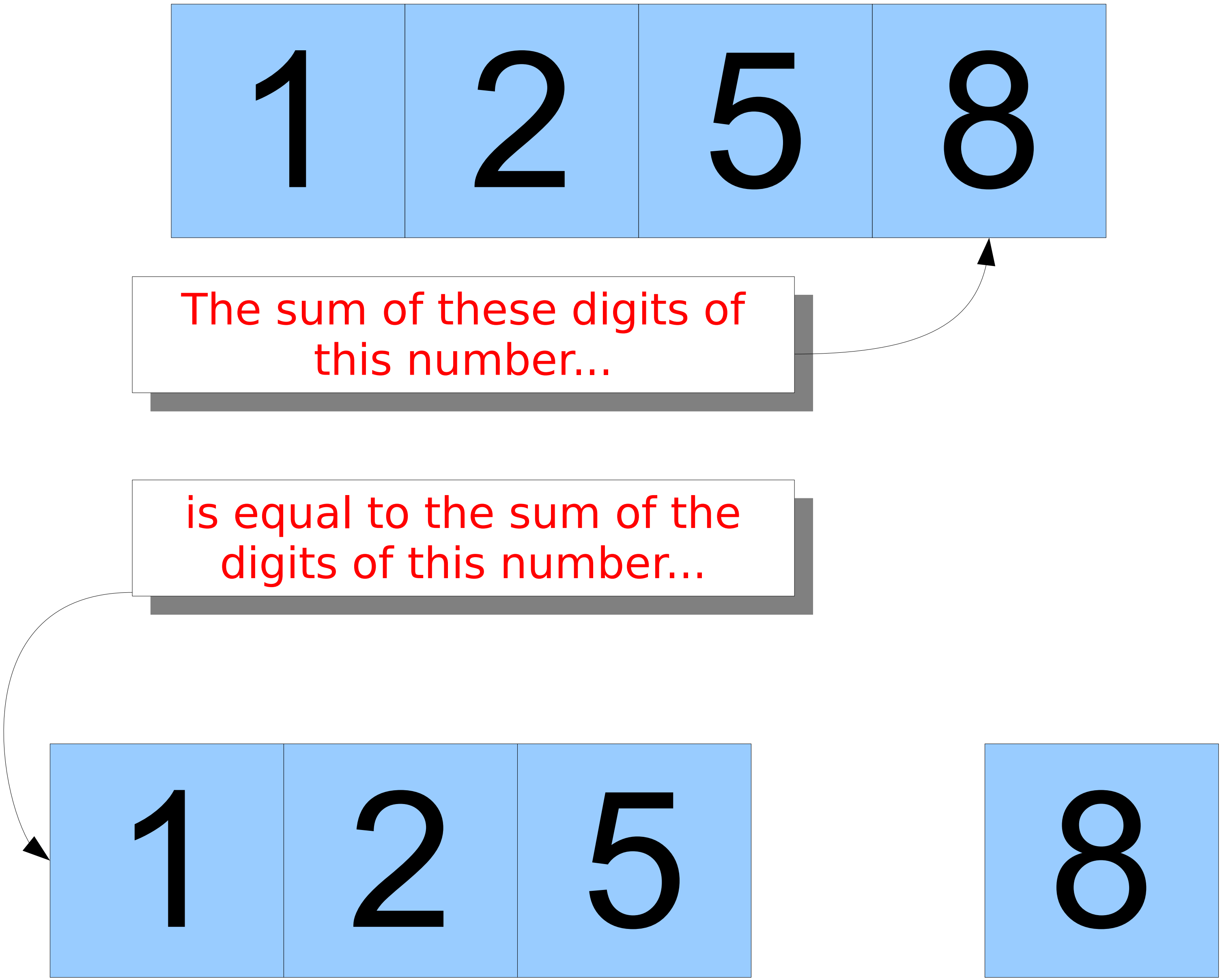
Summing Up Digits

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

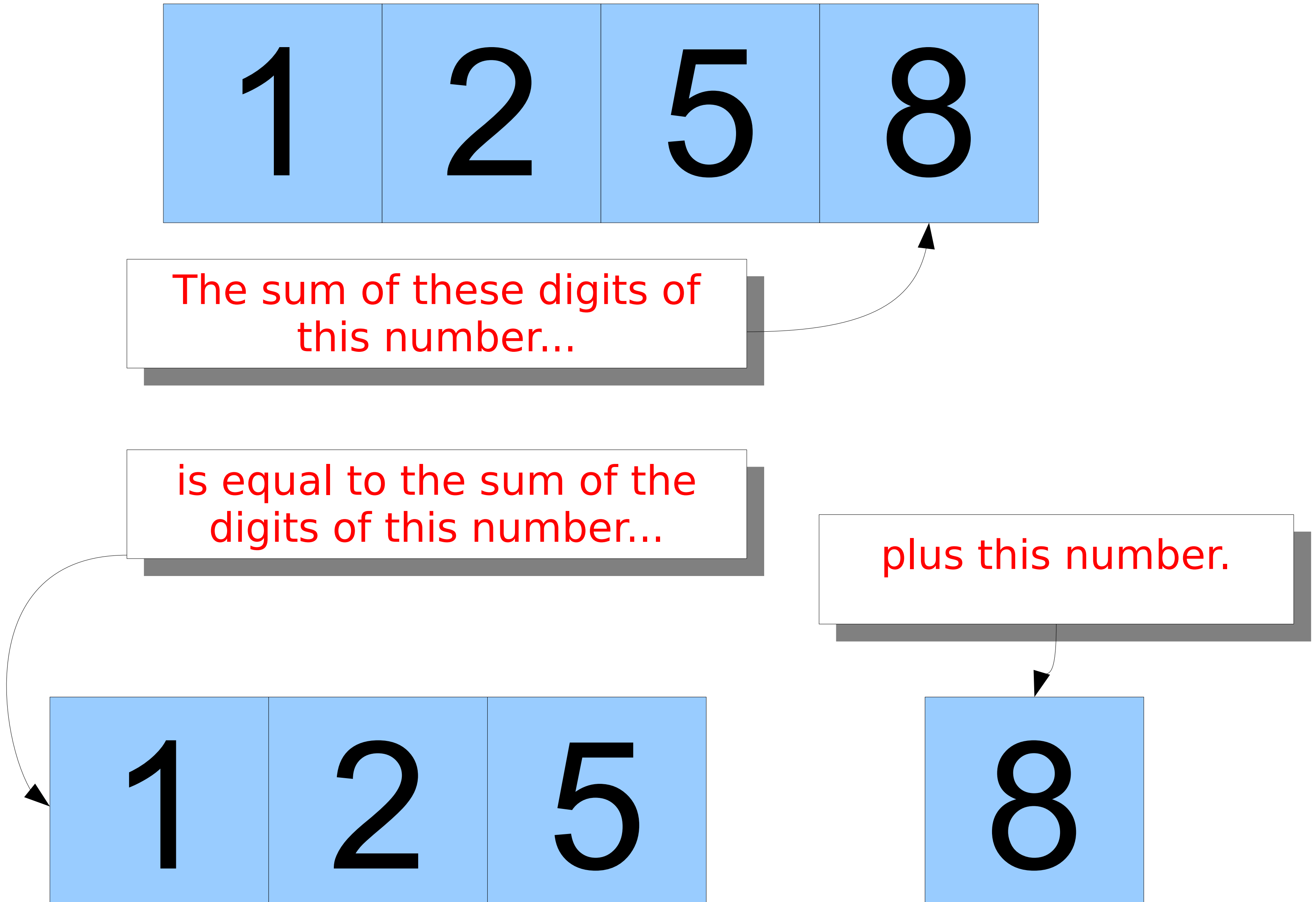
```
int sumOfDigits(int n) {
    int result = 0;
    while (n != 0) {
        result += n % 10;
        n /= 10;
    }
    return result;
}
```

- How would we rewrite this function recursively?

Summing Up Digits



Summing Up Digits



digital-roots.cpp
(On Computer)

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 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:
  - If the problem is simple, solve it directly.
  - If not, solve a smaller version of the same problem.

Recursion vs. Iteration

- Any problem solved using *iteration* (for/while loops) can be solved using *recursion*
- All the recursive solutions we've covered today can be solved equally well using iteration
  - This is to help us feel more comfortable with recursion
  - When the choice is available, iteration is preferred to recursion
- Soon we'll start covering problems that can only be solved using recursion

Today

- Getting Started in C++
- Thinking Recursively
- Style Gameshow
- Parameter Passing and Common Mistakes

Style Gameshow

- Style is a very important part of programming.
  - In the real world, other people need to be able to read your code!
- Guess what I don't like about the style of the code and get a prize!

Bad Style #1

```
int spork(int x, int y) {
   int p = Mumbo(y);

   int pp = Jumbo(x);

   if (p*pp > 0) {

      return Jabba(p);

   }

   return Jabba(pp);

}
```

Bad Style #1

```
int spork(int x, int y) {
   int p = Mumbo(y);
   int pp = Jumbo(x);
   if (p*pp > 0) {
      return Jabba(p);
   }
   return Jabba(pp);
}
```

I have no clue what is going on in this function!!!

Good Style #1

```
int calculateAreaOfRectangle(int width, int height) {
   return width*height
}
```

Bad Style #2

```
void printPrimeNumbers() {
   for (int i = 0; i < 20; i++) {

      if (isPrime(i)) {

         cout << i << endl;

      }

   }

}
```

Bad Style #2

```
void printPrimeNumbers() {
    for (int i = 0; i < 20; i++) {
        if (isPrime(i)) {
            cout << i << endl;
        }
    }
}
```

Magic Number!

Better Style #2

```
const int kMaxPrime = 20;

int main() {

    printPrimeNumbers();

}

void printPrimeNumbers() {
   for (int i = 0; i < kMaxPrime; i++) {

      if (isPrime(i)) {

         cout << i << endl;

      }

   }

}
```

Best Style #2

```
const int kMaxPrime = 20;

int main() {

    printPrimeNumbers(kMaxPrime);

}

void printPrimeNumbers(int maxPrime) {
   for (int i = 0; i < maxPrime; i++) {

      if (isPrime(i)) {

         cout << i << endl;

      }

   }

}
```

Bad Style #3

```
if (isWord == true) {
   return true;
} else {
   return false;
}
```

Bad Style #3

```
if (isWord == true) {
    return true;
} else {
    return false;
}
```

Redundant boolean check

Better Style #3

```
if (isWord) {
    return true;
} else {
    return false;
}
```

Best Style #3

```
if (isWord) {
    return true;
} else {
    return false;
}
return isWord;
```

Bad Style #4

```
const int kSumMax = 10;
int sum;
int main() {
    sum = 0;
    for (int i = 0; i < kSumMax; i++) {
        sum += i;
    }
    cout << “Sum:” << sum;
    return 0;
}
```

Next Time

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