Strings in C++

Recap from Last Time

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);
    }
}
```

New Stuff!

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 step***.

Thinking Recursively

```
if (The problem is very simple) {
    Directly solve the problem.
    Return the solution.
} else {
    Split the problem 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.
}
```

These simple cases are called **base cases**.

These are the **recursive cases**.

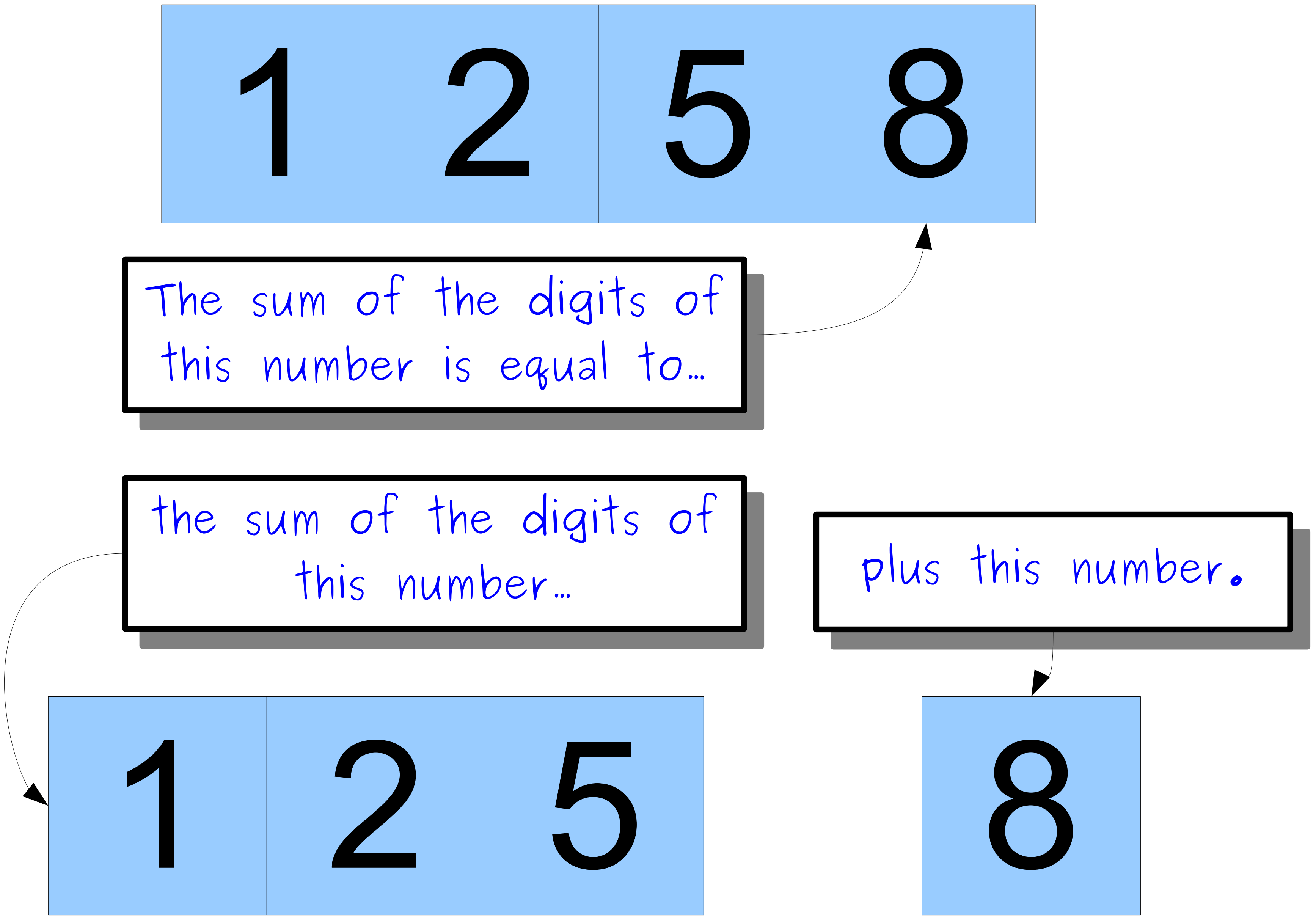
Summing Up Digits

- On Wednesday, we wrote this function to sum up the digits of a nonnegative integer:

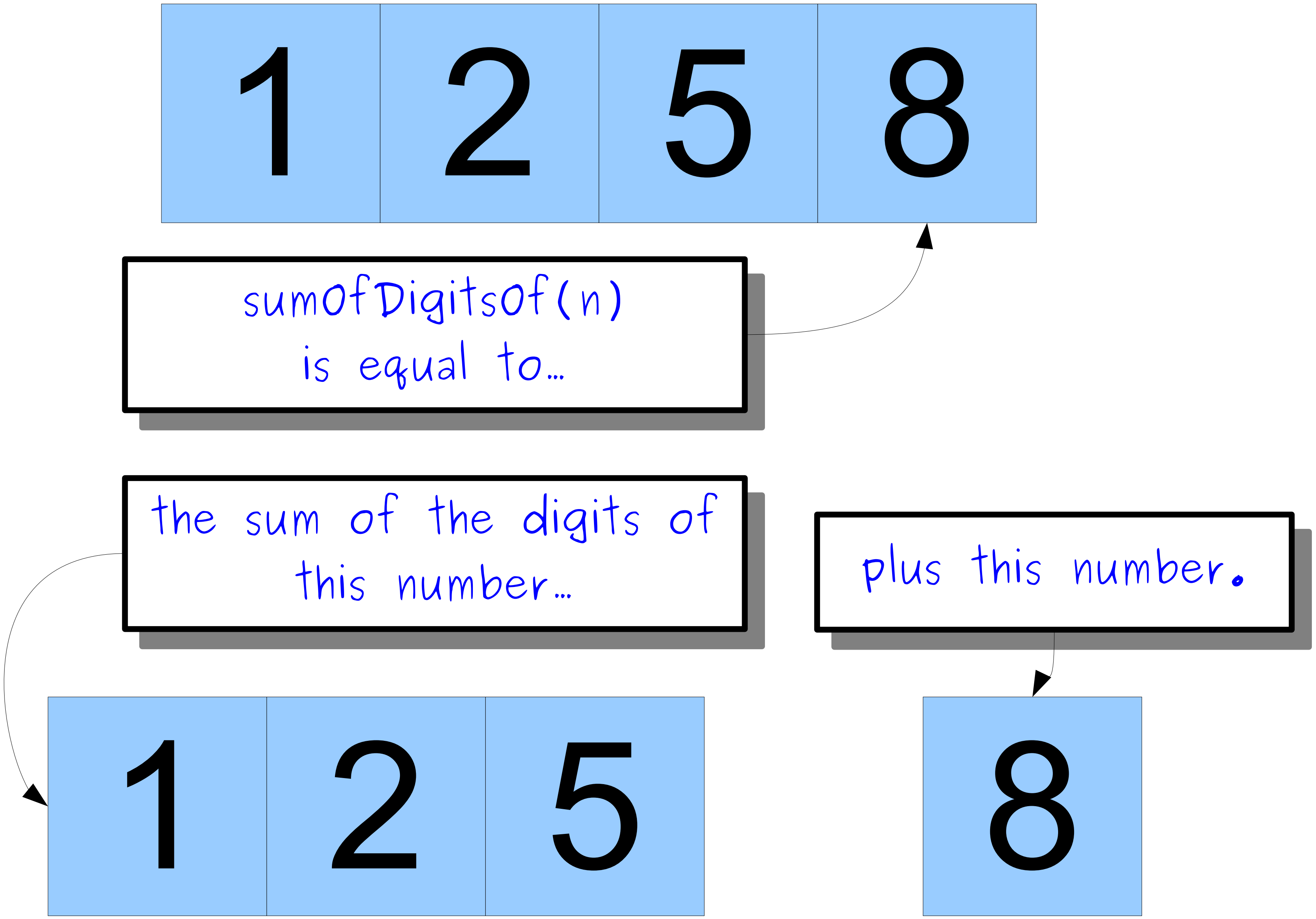
```
int sumOfDigitsOf(int n) {
    int result = 0;

    while (n > 0) {
        result += (n % 10);
        n /= 10;
    }

    return result;

}
```

- Let’s rewrite this function recursively!

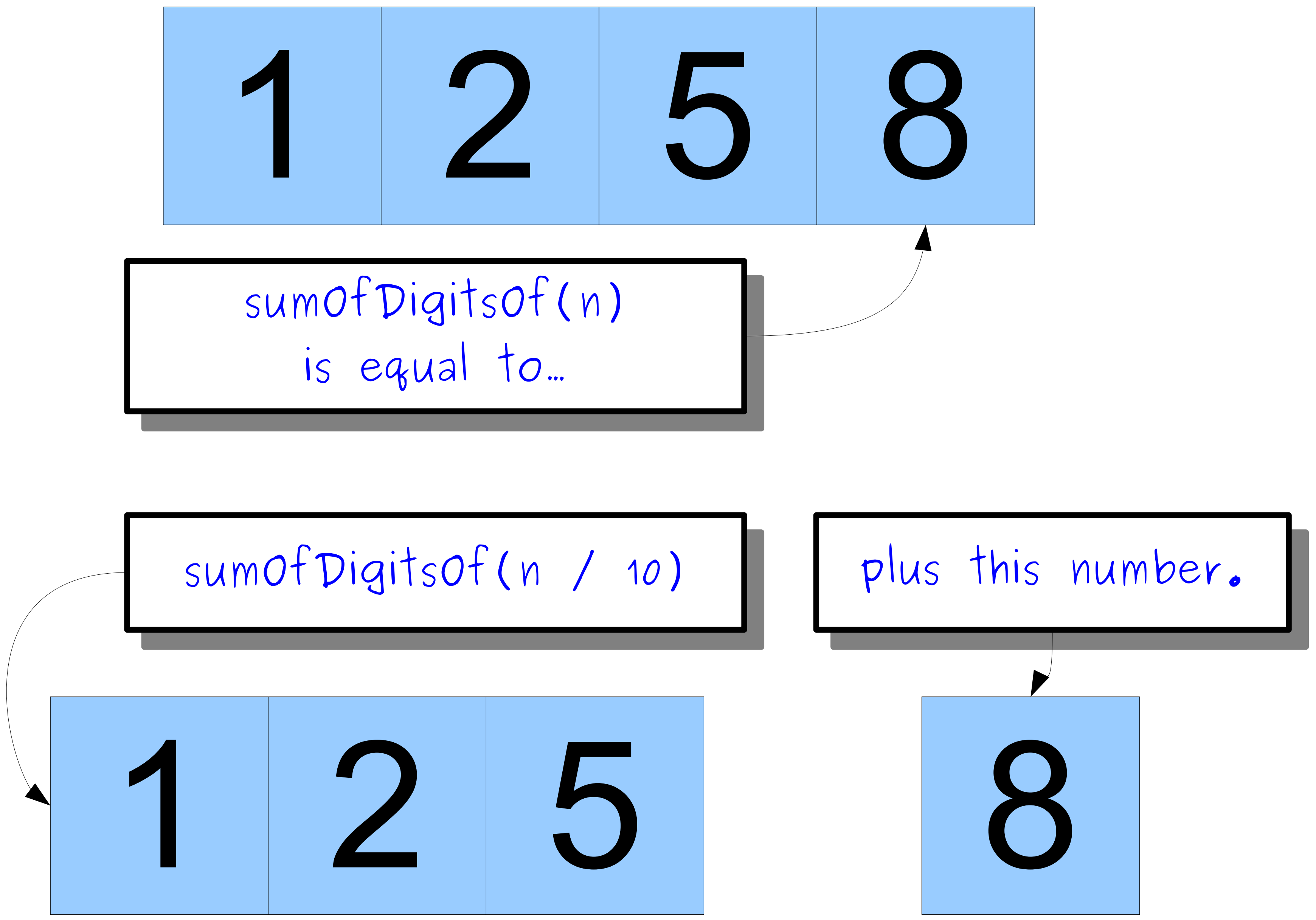
Summing Up Digits


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

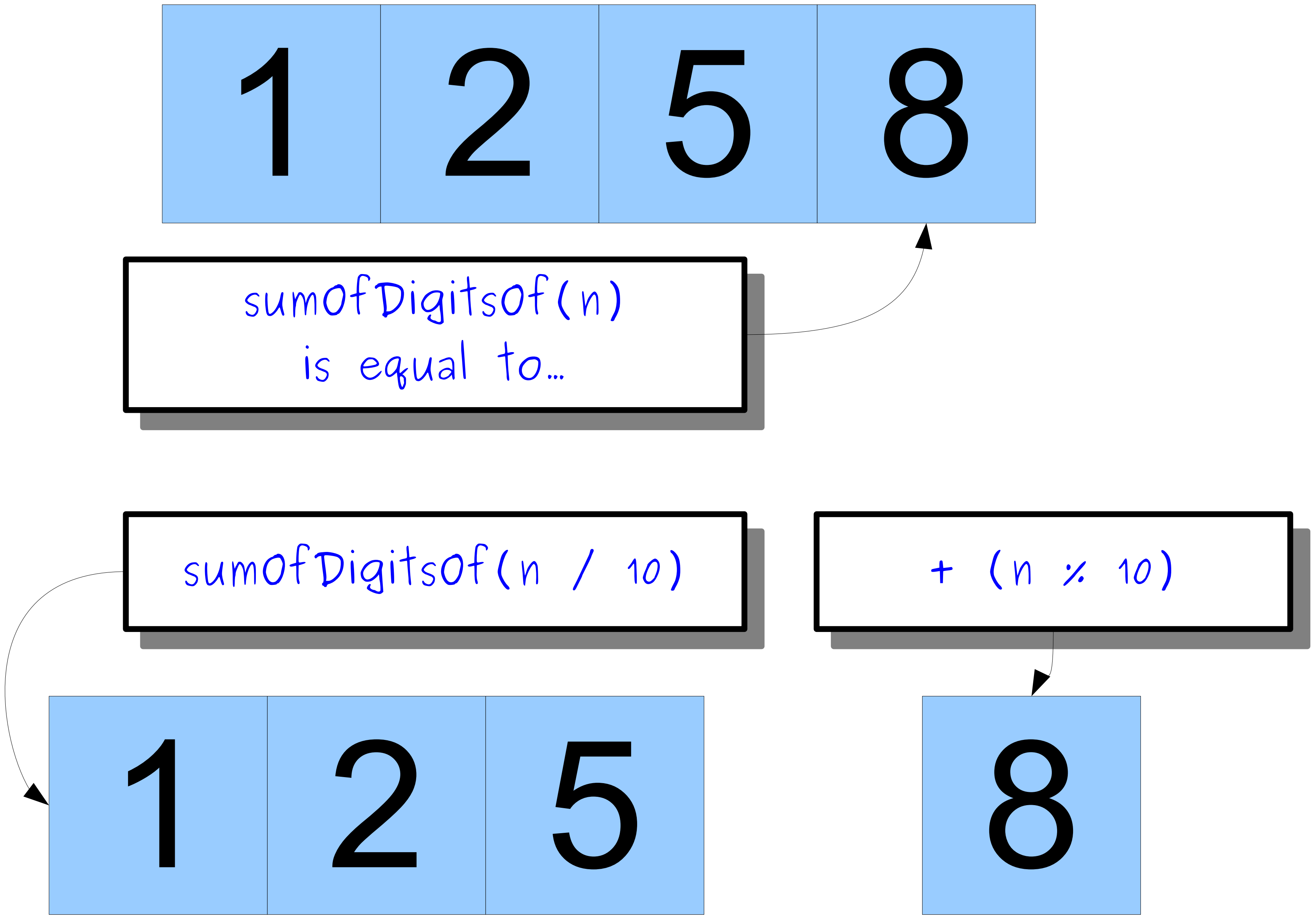
Summing Up Digits



Summing Up Digits



Summing Up Digits



Thinking Recursively

```
if (The problem is very simple) {
    Directly solve the problem.
    Return the solution.
} else {
    Split the problem 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.
}
```

These simple cases are called base cases.

These are the recursive cases.

Tracing the Recursion

```
int main() {
   int sum = sumOfDigitsOf(137);
   cout << "Sum is " << sum << endl;
}
```

Tracing the Recursion

```
int main() {
    int sum = sumOfDigitsOf(137);
    cout << "Sum is " << sum << endl;
}
```

Tracing the Recursion

```
int main() {

}
```

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

int n **137**

Tracing the Recursion

```
int main() {

}
```

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

int n 137

Tracing the Recursion

```
int main() {

}
```

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

int n: 137

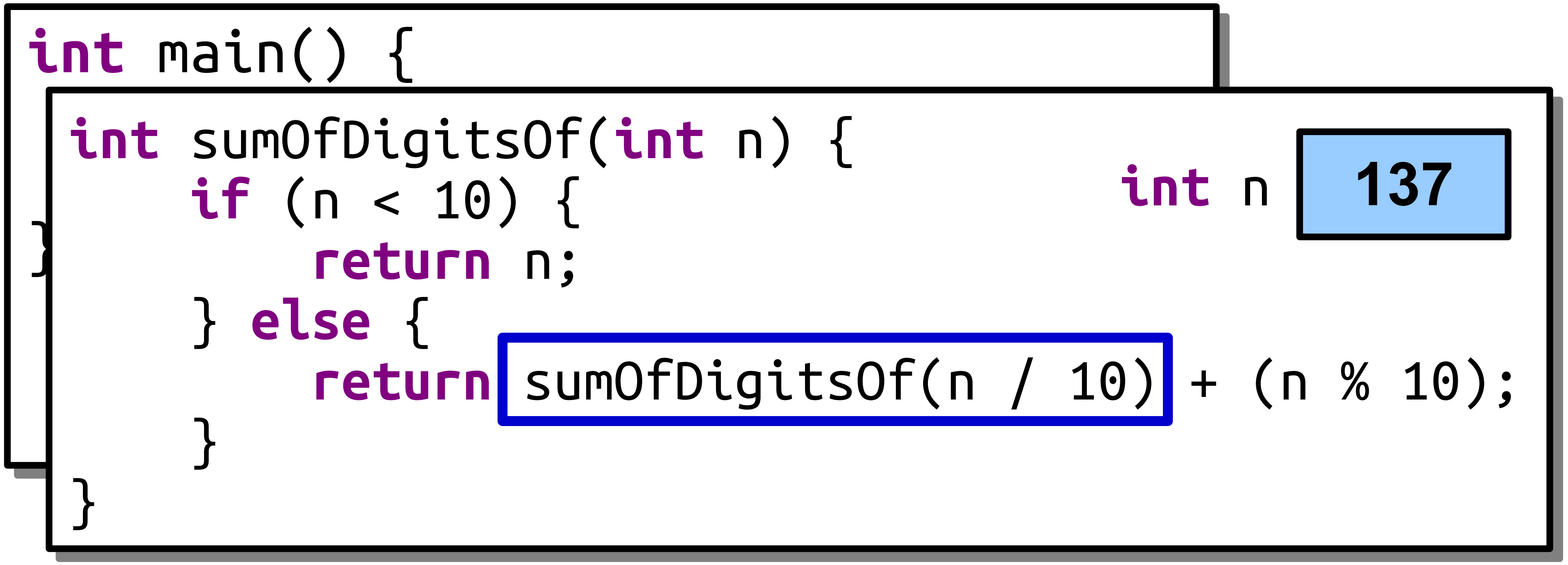
Tracing the Recursion

```
int main() {

}
```

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

int n $\boxed{137}$

Tracing the Recursion



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

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

Tracing the Recursion

```
int main() {

}
```

```
int sumOfDigitsOf(int n) {

}
```

```
int sumOfDigitsOf(int n) {                          int n   13
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

Tracing the Recursion

```
int main() {

}
```

```
int sumOfDigitsOf(int n) {

}
```

```
int sumOfDigitsOf(int n) {                    int n  13
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

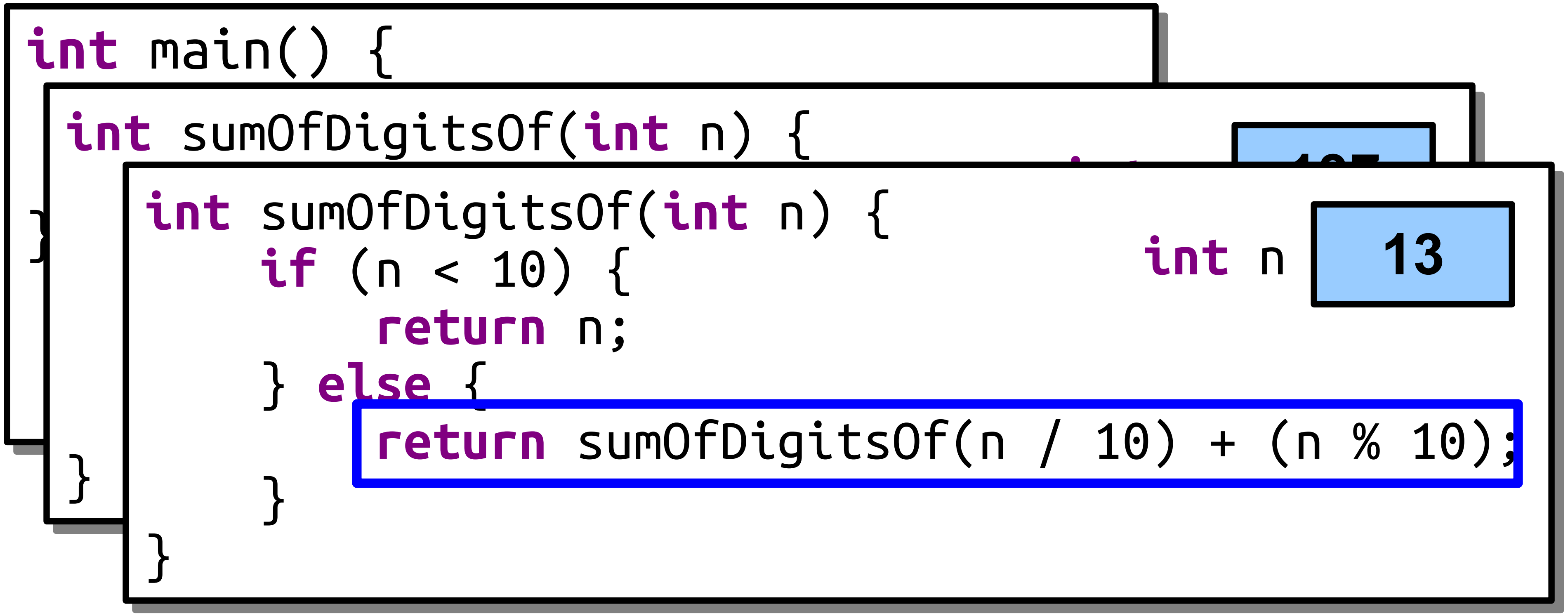
Tracing the Recursion

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

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {                          int n  13
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

Tracing the Recursion

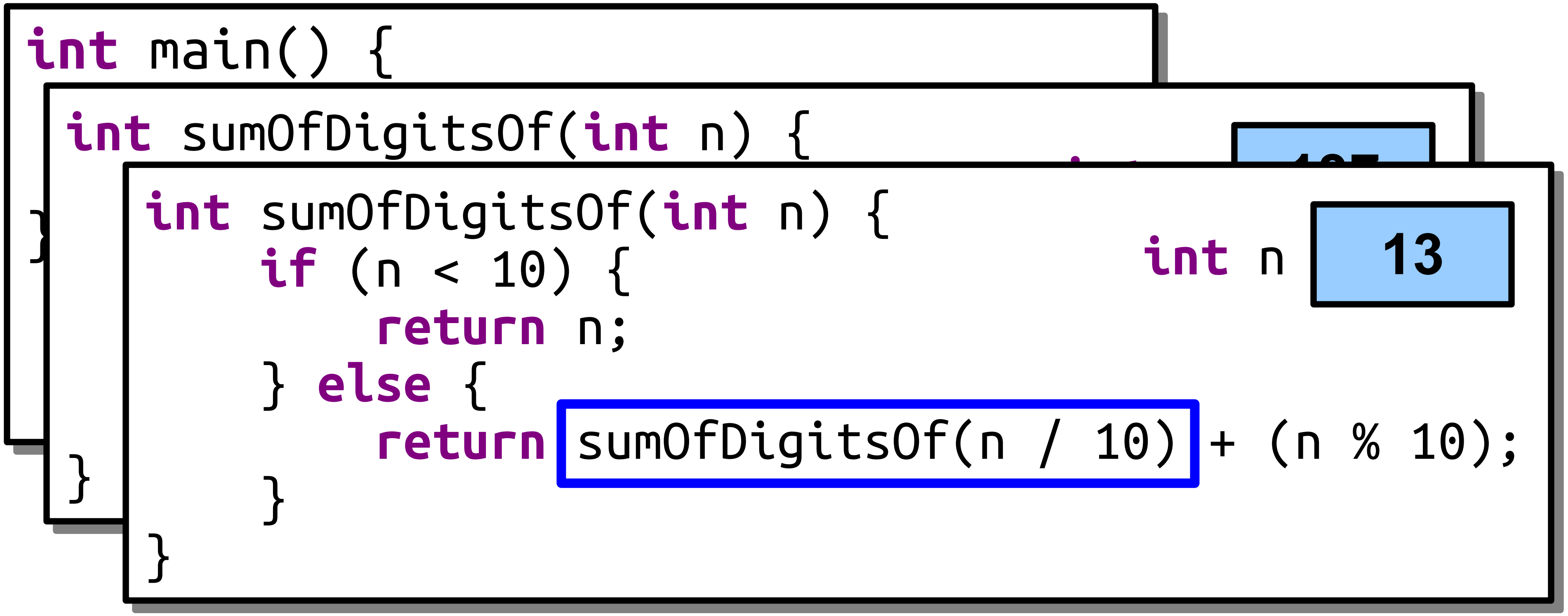


```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

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

Tracing the Recursion

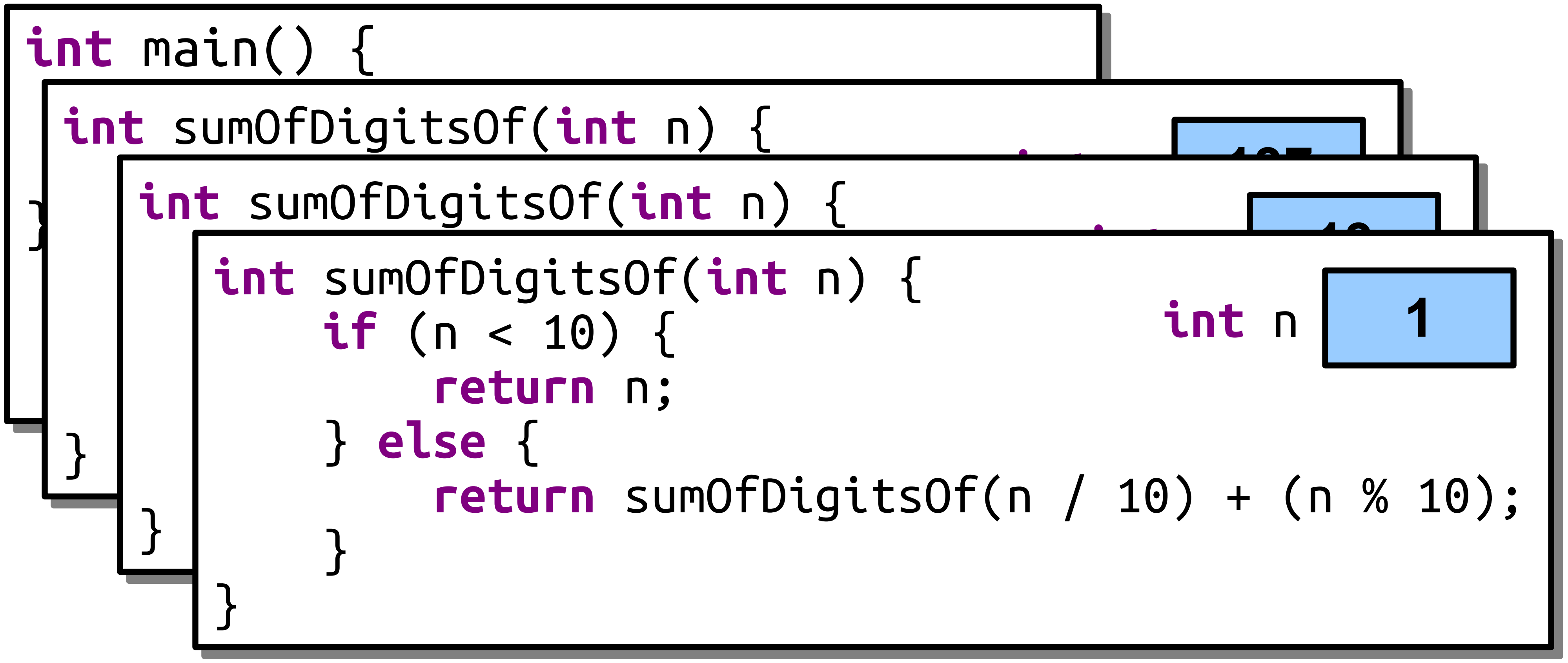


```
int main() {
```

```
int sumOfDigitsOf(int n) {
```

```
int sumOfDigitsOf(int n) {                    int n  13
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

Tracing the Recursion



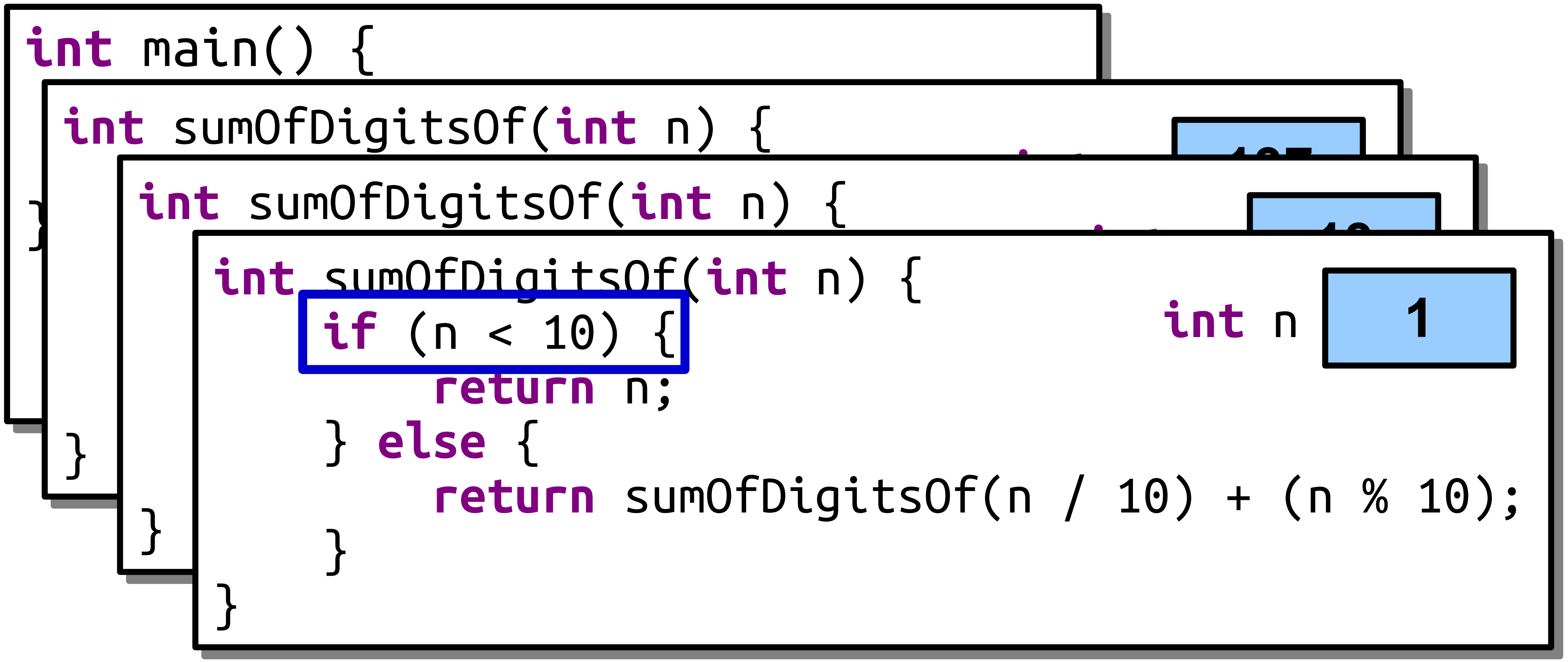
```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {                         int n   1
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

Tracing the Recursion



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

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {                           int n  1
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

Tracing the Recursion

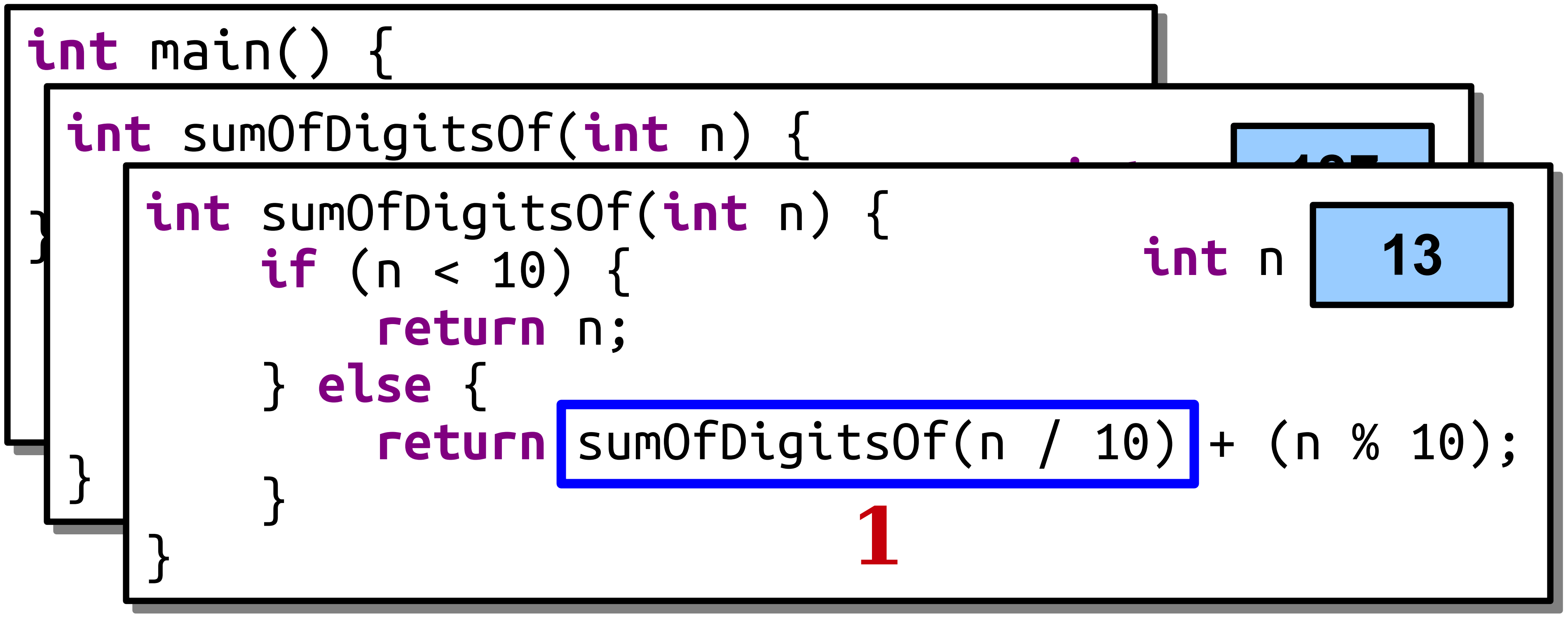
```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {                     int n  1
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

Tracing the Recursion

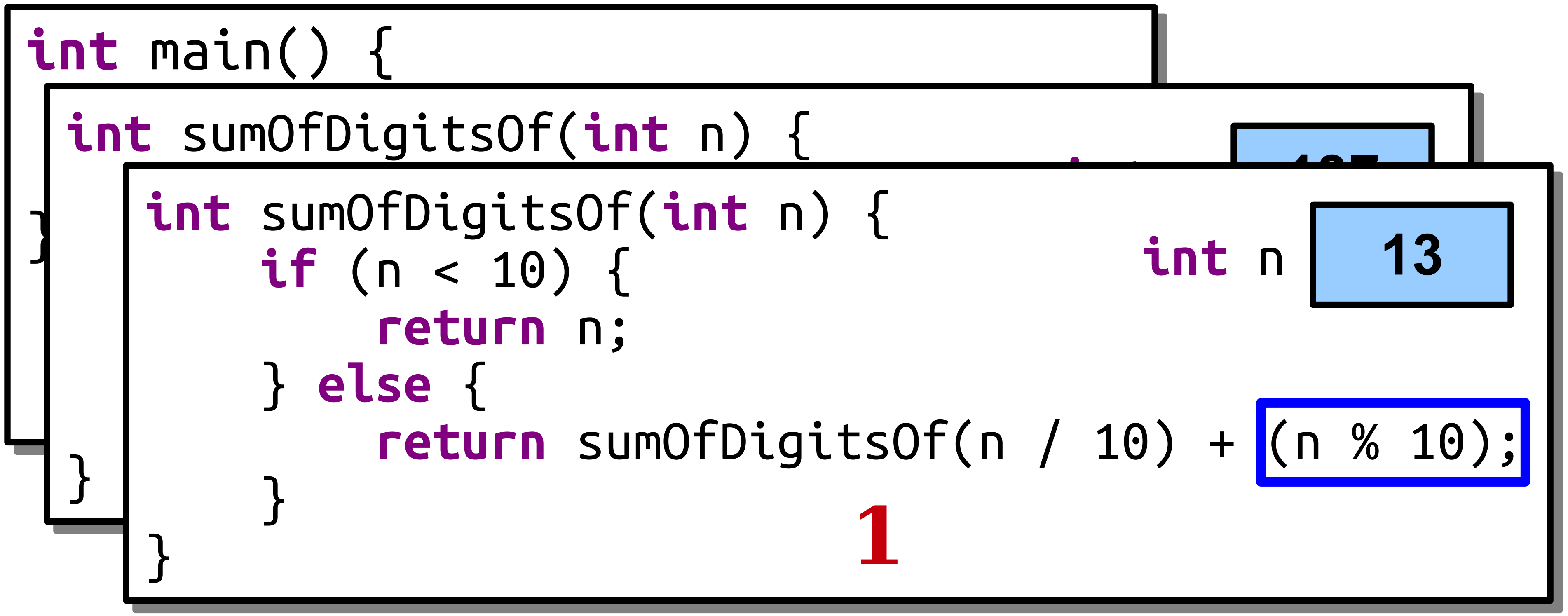


```
int main() {
}
```

```
int sumOfDigitsOf(int n) {
}
```

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

Tracing the Recursion



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

```
int sumOfDigitsOf(int n) {
}
```

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

Tracing the Recursion

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

```
int sumOfDigitsOf(int n) {
}
```

```
int sumOfDigitsOf(int n) {                          int n  [ 13 ]
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }
}
```

1 + 3

Tracing the Recursion

```
int main() {

}
```

```
int sumOfDigitsOf(int n) {

}
```

```
int sumOfDigitsOf(int n) {                         int n  13
    if (n < 10) {
        return n;
    } else {
        return sumOfDigitsOf(n / 10) + (n % 10);
    }                                   4
}
```

Tracing the Recursion

```
int main() {

}
```

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

int n $\boxed{137}$

4

Tracing the Recursion

```
int main() {

}
```

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

int n: 137

4

Tracing the Recursion

```
int main() {

}
```

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

int n 137

4 + 7

Tracing the Recursion

```
int main() {

}
```

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

int n $\boxed{137}$

11

Tracing the Recursion

```
int main() {
    int sum = sumOfDigitsOf(137);
    cout << "Sum is " << sum << endl;
}
```

11

Thinking Recursively

```
if (The problem is very simple) {
    Directly solve the problem.
    Return the solution.
} else {
    Split the problem 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.
}
```

These simple cases are called **base cases**.

These are the **recursive cases**.

Example: ***Digital Roots***

Digital Roots

- The ***digital root*** is the number you get by repeatedly summing the digits of a number until you're down to a single digit.
- What is the digital root of 5?
- What is the digital root of 27?
- What is the digital root of 137?

Digital Roots

- The ***digital root*** is the number you get by repeatedly summing the digits of a number until you're down to a single digit.
- What is the digital root of 5?
  - 5 is a single digit, so the answer is 5.
- What is the digital root of 27?
- What is the digital root of 137?

Digital Roots

- The ***digital root*** is the number you get by repeatedly summing the digits of a number until you're down to a single digit.
- What is the digital root of 5?
  - 5 is a single digit, so the answer is 5.
- What is the digital root of 27?
  - 2 + 7 = 9.
- What is the digital root of 137?

Digital Roots

- The ***digital root*** is the number you get by repeatedly summing the digits of a number until you’re down to a single digit.
- What is the digital root of 5?
  - 5 is a single digit, so the answer is 5.
- What is the digital root of 27?
  - 2 + 7 = 9.
  - The answer is 9.
- What is the digital root of 137?

Digital Roots

- The ***digital root*** is the number you get by repeatedly summing the digits of a number until you're down to a single digit.
- What is the digital root of 5?
  - 5 is a single digit, so the answer is 5.
- What is the digital root of 27?
  - 2 + 7 = 9.
  - The answer is 9.
- What is the digital root of 137?
  - 1 + 3 + 7 = 11.

Digital Roots

- The ***digital root*** is the number you get by repeatedly summing the digits of a number until you're down to a single digit.
- What is the digital root of 5?
  - 5 is a single digit, so the answer is 5.
- What is the digital root of 27?
  - 2 + 7 = 9.
  - The answer is 9.
- What is the digital root of 137?
  - 1 + 3 + 7 = 11.
  - 1 + 1 = 2.

Digital Roots

- The ***digital root*** is the number you get by repeatedly summing the digits of a number until you're down to a single digit.
- What is the digital root of 5?
  - 5 is a single digit, so the answer is 5.
- What is the digital root of 27?
  - 2 + 7 = 9.
  - The answer is 9.
- What is the digital root of 137?
  - 1 + 3 + 7 = 11.
  - 1 + 1 = 2.
  - The answer is 2.

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

Thinking Recursively

```
if (The problem is very simple) {
    Directly solve the problem.
    Return the solution.
} else {
    Split the problem 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.
}
```

These simple cases are called *base cases*.

These are the *recursive cases*.

Time-Out for Announcements!

Section Signups

- Section signups are open right now. They close Sunday at 5PM.
- Sign up for section at

  https://cs198.stanford.edu/

- Click on “CS106 Sections Login,” then choose “Section Signup.”

Assignment 1

- Assignment 0 was due today at the start of class.
- ***Assignment 1: Welcome to C++*** goes out today. It's due on Friday, January 17th at the start of class.
    - Play around with C++ and the Stanford libraries!
    - Get some practice with recursion!
    - Explore the debugger!
    - See some pretty pictures! 😃
- We recommend making slow and steady progress on this assignment throughout the course of the week. There's a recommended timetable on the front page of the handout.
- We've posted two handouts online. We strongly recommend reading over them before starting.
    - Handout 06: Debugging Your Code
    - Handout 07: Assignment Submission Checklist

Late Periods

- Everyone has ***two*** free “late periods” to use as needed.
- A “late period” is an automatic extension for one *class period* (Monday to Wednesday, Wednesday to Friday, or Friday to Monday).
- If you need an extension beyond late periods, please talk to Katherine. Your section leader cannot grant extensions.

Assignment Grading

- Your coding assignments are graded on both functionality and on coding style.
- The ***functionality score*** is based on correctness.
  - Do your programs produce the correct output?
  - Do they work on all inputs?
  - etc.
- The ***style score*** is based on how well your program is written.
  - Are your programs well-structured?
  - Do you decompose problems into smaller pieces?
  - Do you use variable naming conventions consistently?
  - etc.

Getting Help

Getting Help

- ***LaIR Hours!***
  - Sunday – Thursday, 7PM – 11PM on the first floor of Tresidder Student Union.
  - LaIR hours start this weekend.
- ***Katherine's Office Hours***
  - Tuesdays and Thursdays, 3:00PM – 4:15PM, Gates B02.
- ***Keith's Office Hours***
  - Tuesdays, 10:00AM – 12:00PM, Gates 172,
  - Stop on by! I'm happy to chat about just about anything.

One More Unto the Breach!

Strings in C++

C++ Strings

- C++ strings are represented with the `string` type.
- To use `string`, you must

```
#include <string>
```

  at the top of your program.
- You can get the number of characters in a string by calling either of these functions:

```
str.length()          str.size()
```

- You can read a single character in a string by writing

```
str[index]
```

Strings and Characters

- In C++, there are two types for representing text:
  - The `char` type (***char***acter) represents a single glyph (letter, punctuation symbol, space, etc.)
  - The `string` type represents a sequence of zero or more characters.
- Keep this in mind if you’re transitioning to C++ from Python or JavaScript.
- Check out Chapter 1.5 for more on the difference between strings and characters.

Strings are Mutable

- Unlike strings in Python, Java, and JavaScript, in C++ strings are mutable and their contents can be changed.
- To change an individual character of a string, write

```
str[index] = ch;
```

- To append more text, you can write

```
str += text;
```

- These operations directly change the string itself, rather than making a copy of the string.

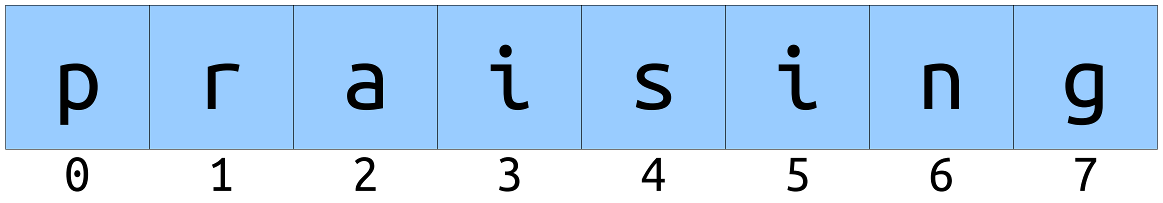
Other Important Differences

- In C++, the == operator can directly be used to compare strings:

```
if (str1 == str2) {
    /* strings match */
}
```

- You can get a substring of a string by calling the `substr` method. `substr` takes in a start position and optional *length* (not an end position!)

```
string allButFirstChar    = str.substr(1);
```



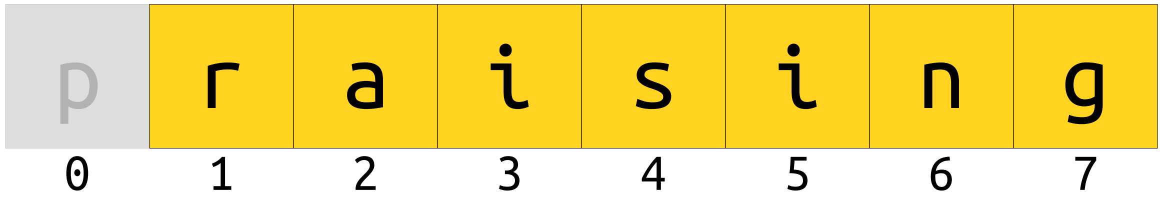
Other Important Differences

- In C++, the == operator can directly be used to compare strings:

```
if (str1 == str2) {
    /* strings match */
}
```

- You can get a substring of a string by calling the `substr` method. `substr` takes in a start position and optional *length* (not an end position!)

```
string allButFirstChar    = str.substr(1);
```



Other Important Differences

- In C++, the == operator can directly be used to compare strings:

```
if (str1 == str2) {
    /* strings match */
}
```

- You can get a substring of a string by calling the `substr` method. `substr` takes in a start position and optional *length* (not an end position!)

```
string allButFirstChar    = str.substr(1);
string allButFirstAndLast = str.substr(1, str.length() - 2);
```

p	r	a	i	s	i	n	g
0	1	2	3	4	5	6	7

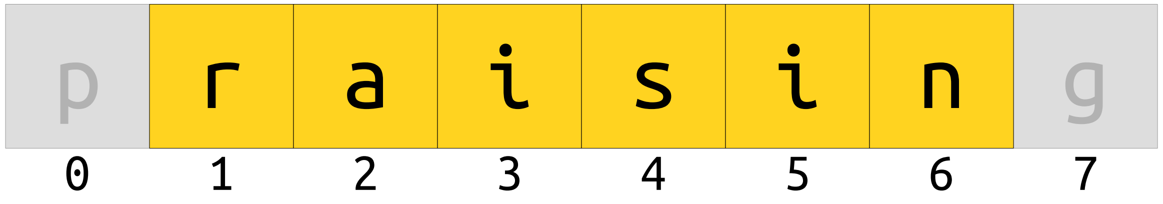
Other Important Differences

- In C++, the == operator can directly be used to compare strings:

```
if (str1 == str2) {
    /* strings match */
}
```

- You can get a substring of a string by calling the `substr` method. `substr` takes in a start position and optional *length* (not an end position!)

```
string allButFirstChar    = str.substr(1);
string allButFirstAndLast = str.substr(1, str.length() - 2);
```



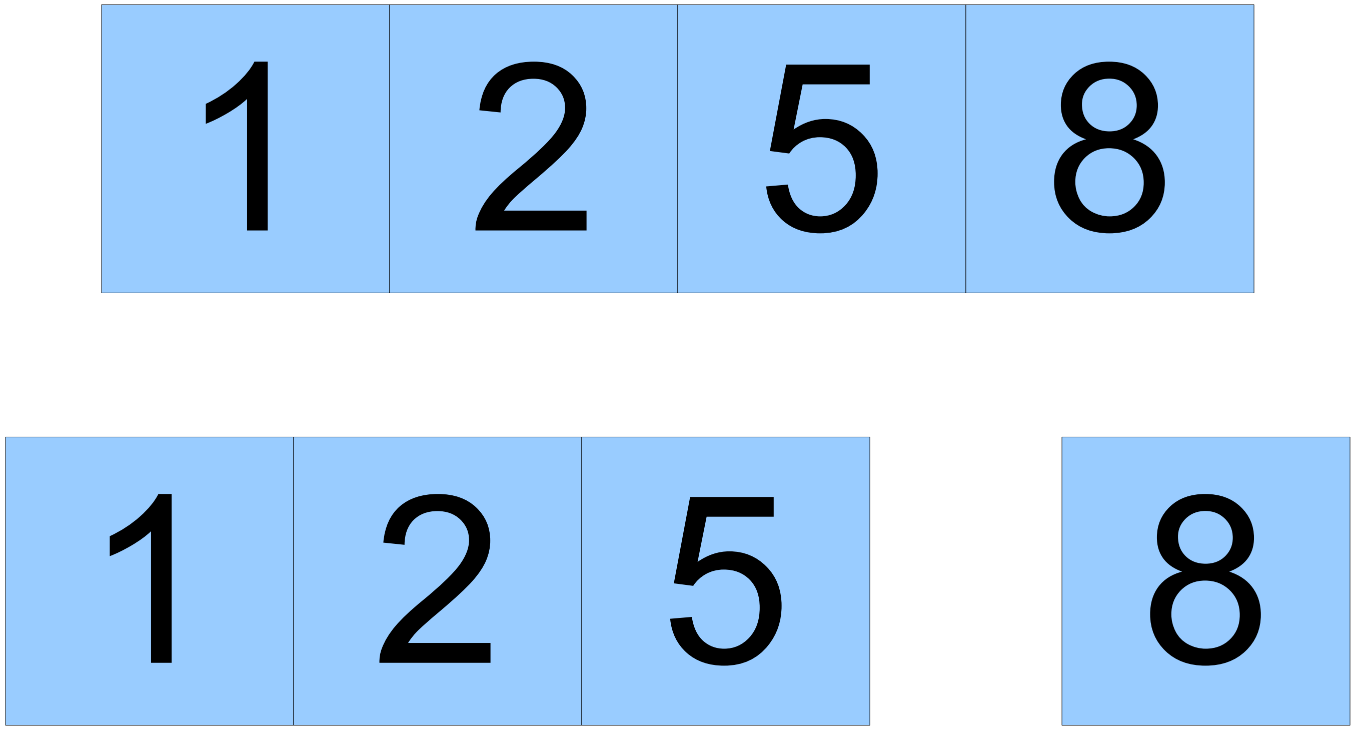
Even More Differences

- In Java and JavaScript you can concatenate just about anything with a string.
- In C++, you can only concatenate strings and characters onto other strings.
- Use the `to_string` function to convert things to strings:

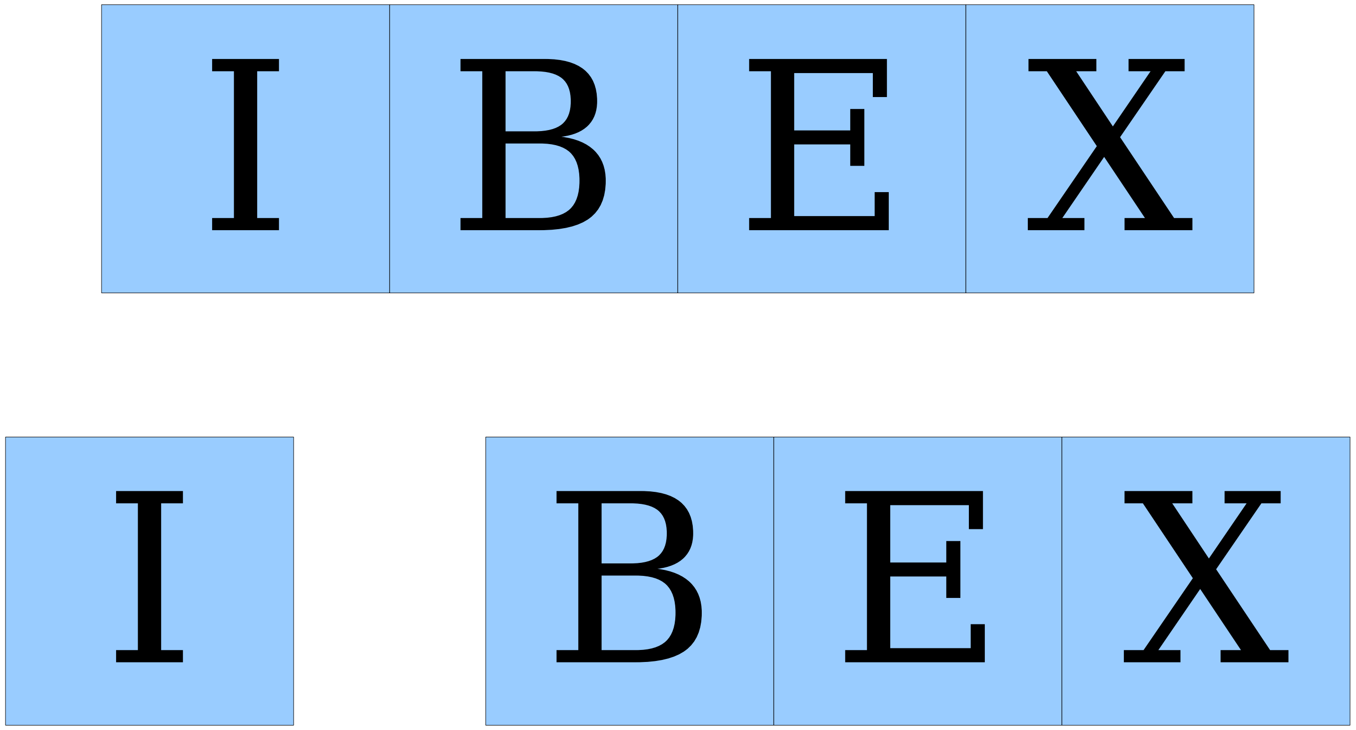
```
string s = "He really likes " + to_string(137);
s += "And also apparently " + to_string(2.718);
```

Recursion and Strings

Thinking Recursively



Thinking Recursively

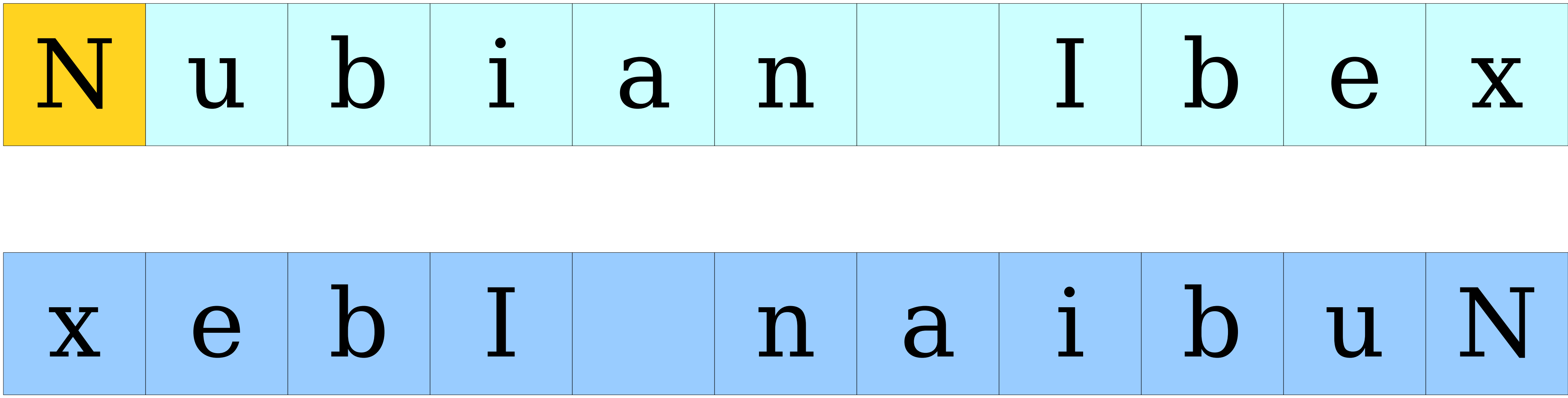


Reversing a String

N	u	b	i	a	n		I	b	e	x

x	e	b	I		n	a	i	b	u	N

Reversing a String

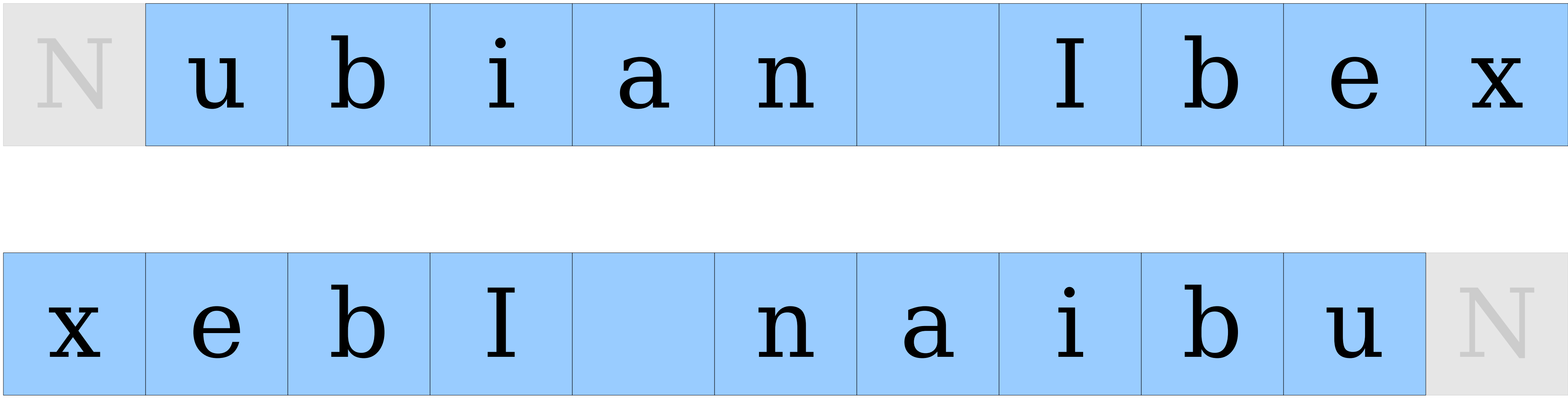


Reversing a String

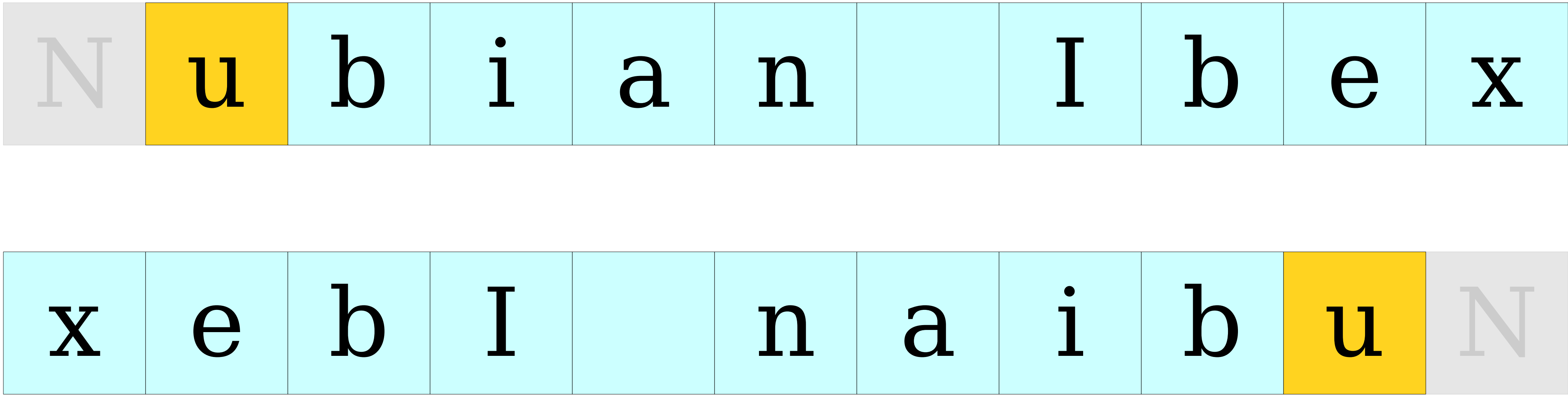
N	u	b	i	a	n		I	b	e	x

x	e	b	I		n	a	i	b	u	N

Reversing a String



Reversing a String



Reversing a String Recursively

```
reverseOf("TOP")
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP")
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") =  reverseOf("P") + O
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") =  reverseOf("P") + O

reverseOf("P")
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") =   reverseOf("P") + O

reverseOf("P") =    reverseOf("") + P
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") =  reverseOf("P") + O

reverseOf("P") =   reverseOf("") + P

reverseOf("") = ""
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") =  reverseOf("P") + O

reverseOf("P") =            "" + P

reverseOf("") = ""
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") =    reverseOf("P") + O

reverseOf("P") =              P

reverseOf("") = ""
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") =                  P  + O

reverseOf("P") =                   P

reverseOf("") = ""
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") =              PO

reverseOf("P") =               P

reverseOf("") = ""
```

Reversing a String Recursively

```
reverseOf("TOP") =          PO          + T

reverseOf("OP") =           PO

reverseOf("P") =            P

reverseOf("") = ""
```

Reversing a String Recursively

```
reverseOf("TOP") =        POT

reverseOf("OP") =         PO

reverseOf("P") =          P

reverseOf("") = ""
```

Reversing a String Recursively

```
reverseOf("TOP") = reverseOf("OP") + T

reverseOf("OP") =  reverseOf("P") + O

reverseOf("P") =   reverseOf("") + P

reverseOf("") = ""
```

Thinking Recursively

```
if (The problem is very simple) {
    Directly solve the problem.
    Return the solution.
} else {
    Split the problem 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.
}
```

These simple cases are called **base cases**.

These are the **recursive cases**.

Recap from Today

- Recursion works by identifying
  - one or more ***base cases***, simple cases that can be solved directly, and
  - one or more ***recursive cases***, where a larger problem is turned into a smaller one.
- C++ strings have some endearing quirks compared to other languages. Importantly, they’re mutable.
- Recursion is everywhere! And you can use it on strings.

Your Action Items

- ***Read Chapter 3.***
  - This chapter is all about strings and string processing, and it has some real winners.
- ***Start working on Assignment 1.***
  - Aim to complete Stack Overflows and one or two of the recursion problems by Monday.

Next Time

- ***The `Vector` Type***
  - Storing sequences in C++!
- ***Recursion on `Vector`s.***
  - Of course. ☺