Strings in C++

Recap from Last Time

Recursion on Numbers

- Here’s a recursive function that computes *n*!:

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

- Here’s a recursive implementation of a function to compute the sum of the digits of a number:

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

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*.

New Stuff!

Digital Roots Revisited

- Here’s some code to compute the digital root of a number:

```
int digitalRootOf(int n) {
    while (n >= 10) {
        n = sumOfDigitsOf(n);
    }
    return n;
}
```

- How might we write this recursively?

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*.

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

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.

Strings are Mutable

- Unlike Java, JavaScript, and Python strings, C++ strings are mutable and their contents can be modified.
- 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.

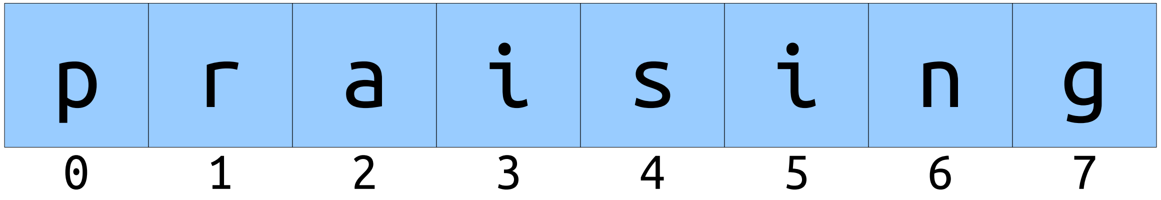
Other Important Differences

- 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);
```



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

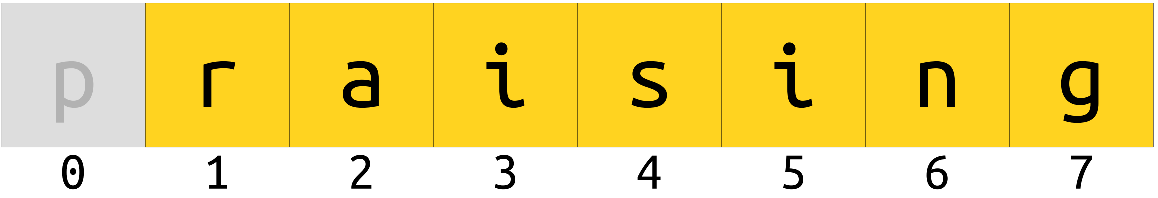
Other Important Differences

- 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);
```



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

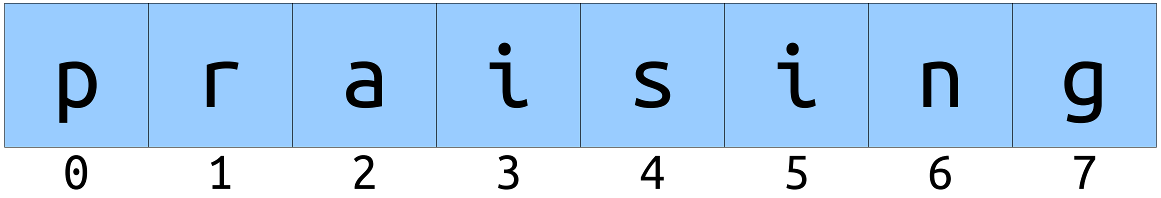
Other Important Differences

- 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);
```



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

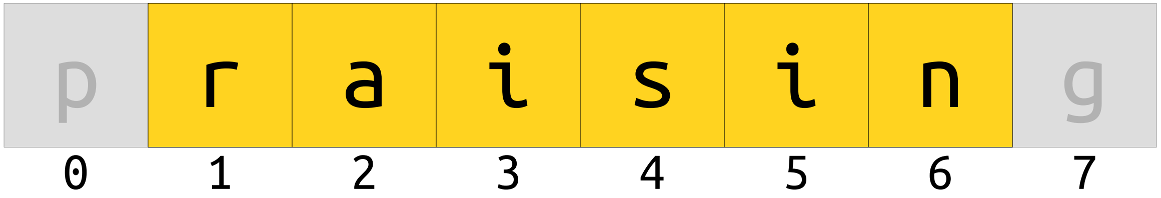
Other Important Differences

- 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);
```



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

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);
```

Time-Out for Announcements!

Migrating to C++ Session

- We'll be holding an extra about migrating from Python or JavaScript to C++. Details below:

Monday, January 14th
7:00PM – 8:30PM
Hewlett 102

- Feel free to stop on by!

Assignment 0

- Assignment 0 was due at the start of today’s lecture.
- Didn’t finish it in time? Don’t worry – you can use your late days to extend the deadline.

Assignment 1

- ***Assignment 1: Welcome to C++*** goes out today. It's due on Friday, January 18th at the start of class.
  - Play around with C++ and the Stanford libraries!
  - Get some practice with recursion.
  - Explore the debugger!
  - Teach the computer to read, sorta. ☺
- 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.

Late Days

- Everyone has ***two*** free “late days” to use as needed.
- A “late day” 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 days, please talk to Kate. 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.

Section Signups

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

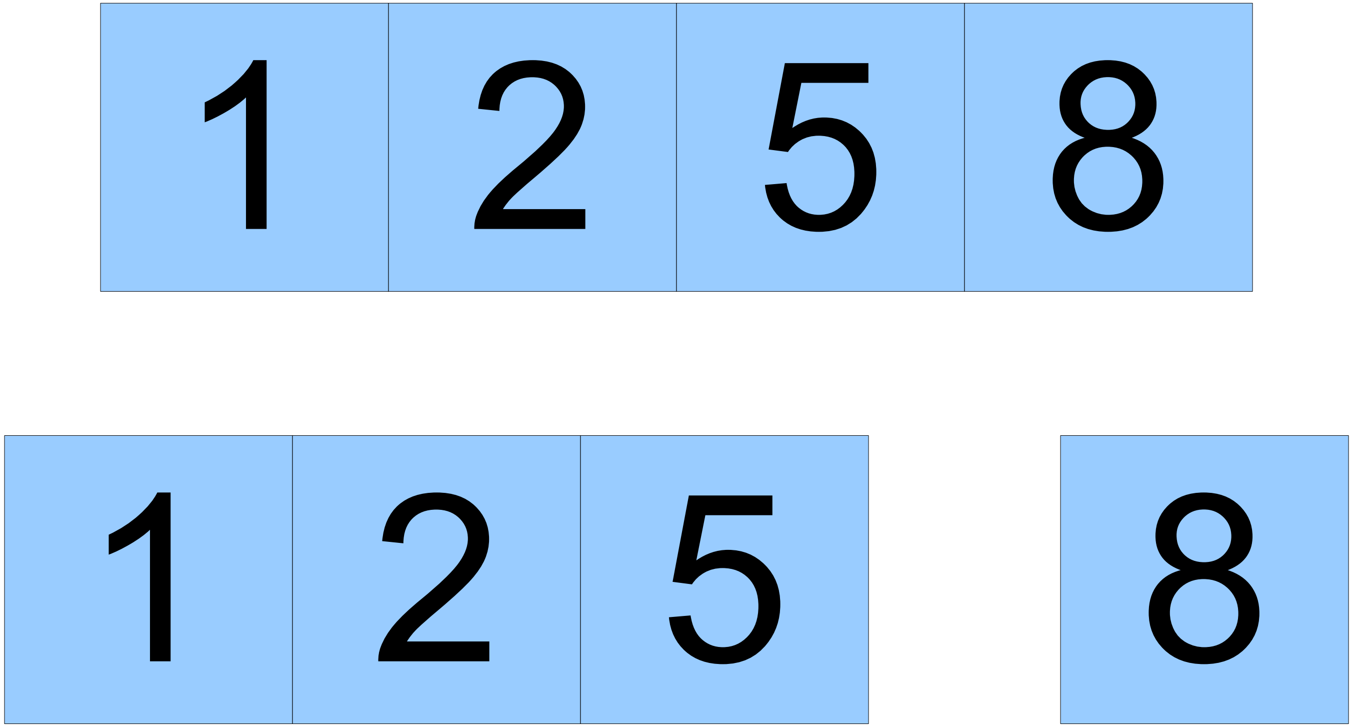
  https://cs198.stanford.edu/

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

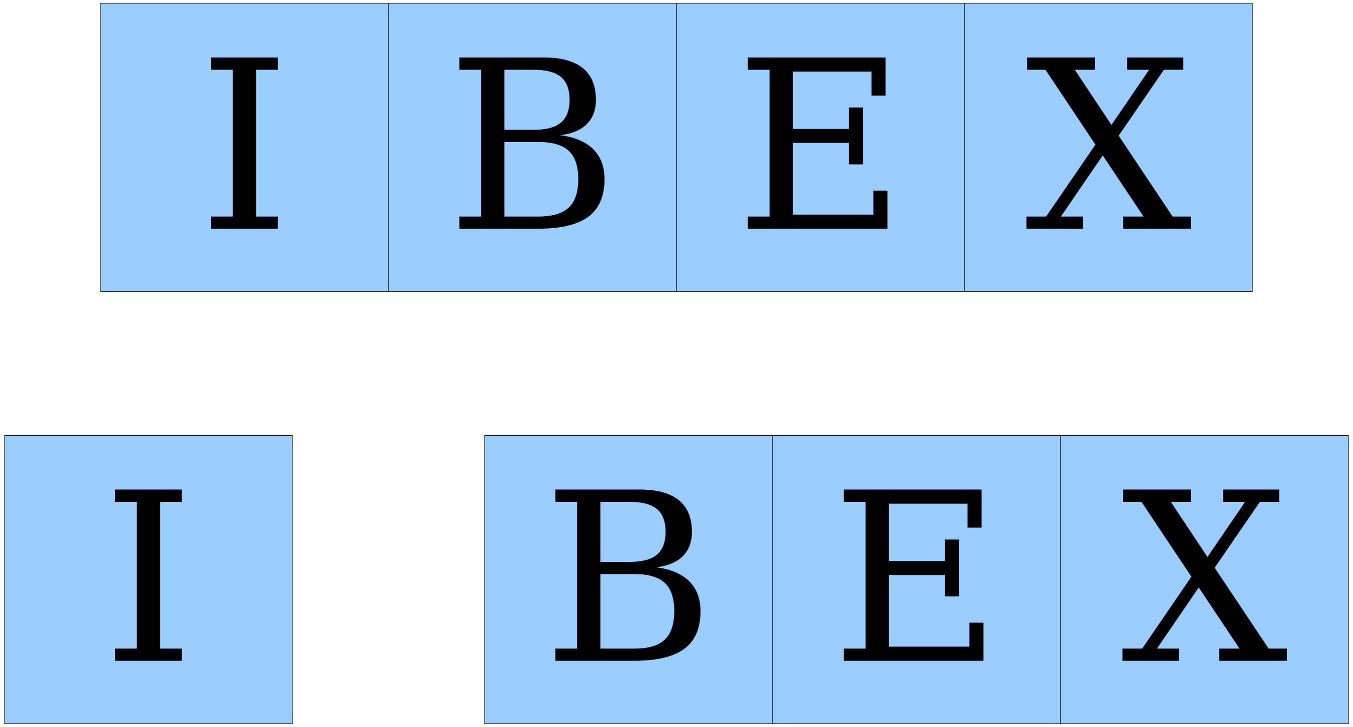
One More Unto the Breach!

Recursion and Strings

Thinking Recursively



Thinking Recursively



Reversing a String

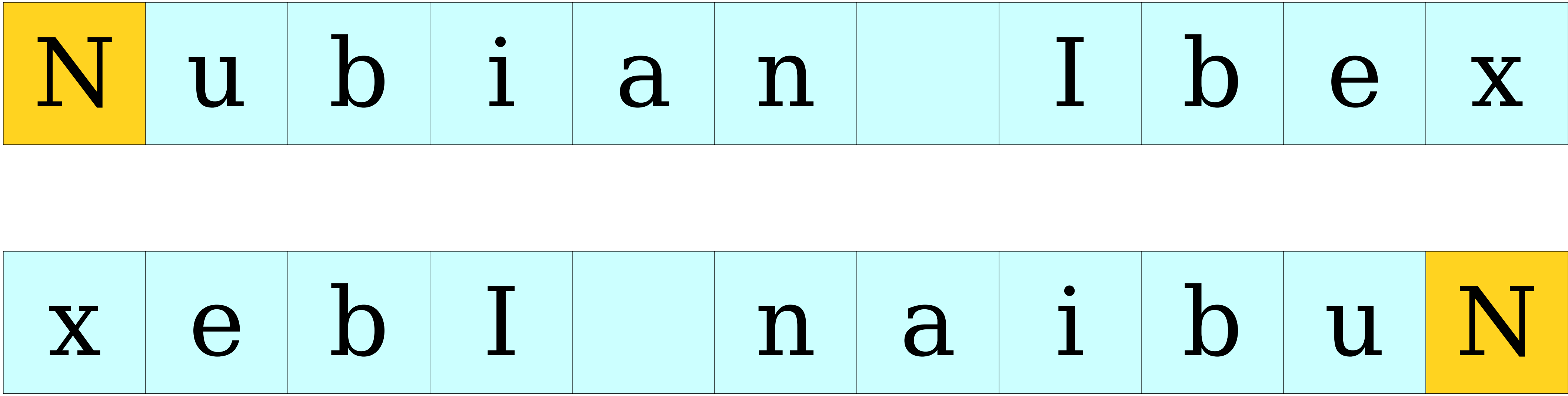
N	u	b	i	a	n		I	b	e	x

x	e	b	I		n	a	i	b	u	N

Reversing a String

N	u	b	i	a	n		I	b	e	x
x	e	b	I		n	a	i	b	u	N

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

N	u	b	i	a	n		I	b	e	x

x	e	b	I		n	a	i	b	u	N

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 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

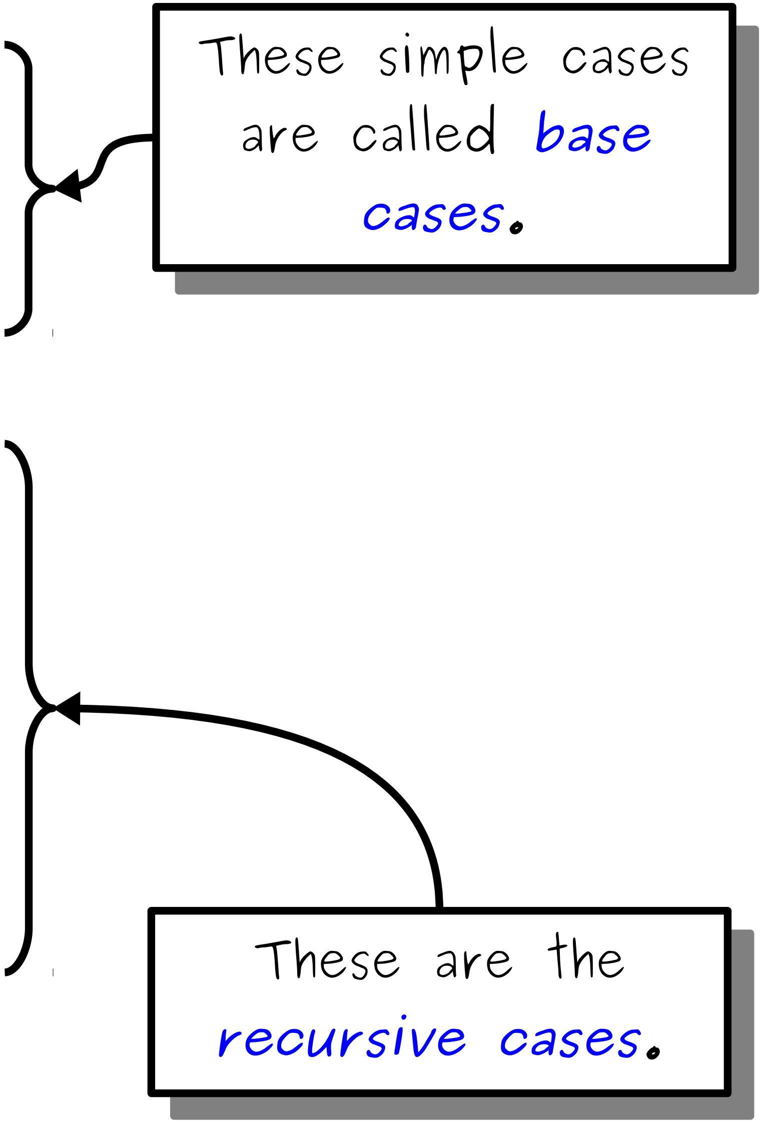
```
reverseOf("TOP") =          PO      + T

reverseOf("OP") =           PO

reverseOf("P") =            P

reverseOf("") = ""
```

Reversing a String Recursively

```
reverseOf("TOP") =            POT

reverseOf("OP") =             PO

reverseOf("P") =              P

reverseOf("") = ""
```

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*.

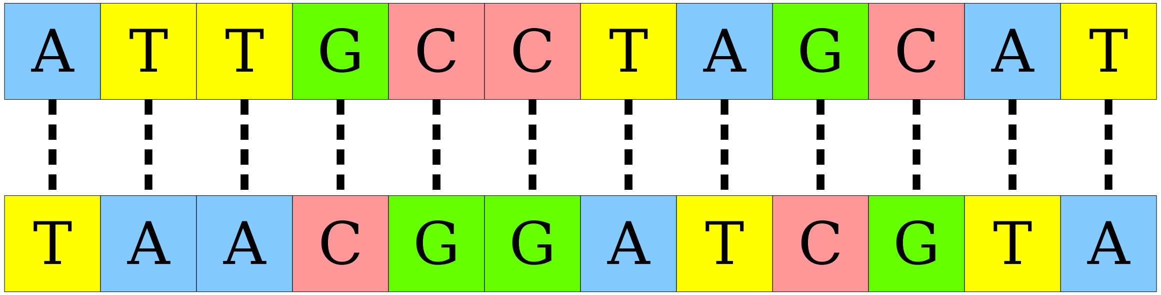
DNA Strands

DNA Strands

- Each strand of DNA consists of a series of ***nucleotides***. There are four nucleotides, abbreviated A, C, G, and T.
- Each nucleotide pairs with another:

  A pairs with T   C pairs with G

- Two strands are called ***complementary*** if each nucleotide in the first pairs with the corresponding nucleotide in the second.



A T T G C C T A G C A T

T A A C G G A T C G T A

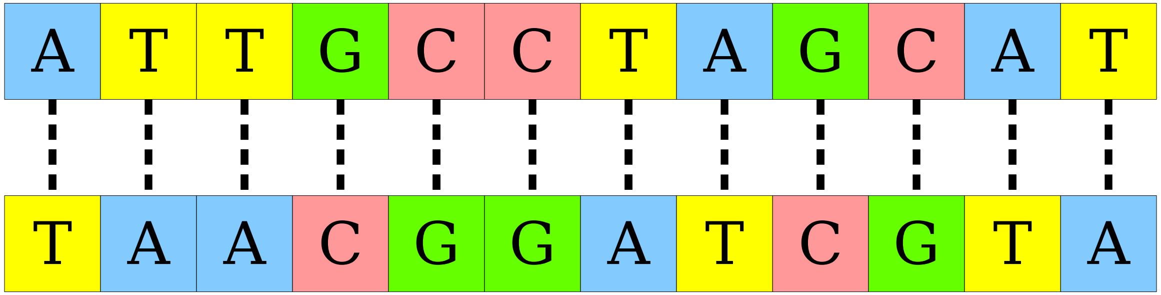
DNA Strands

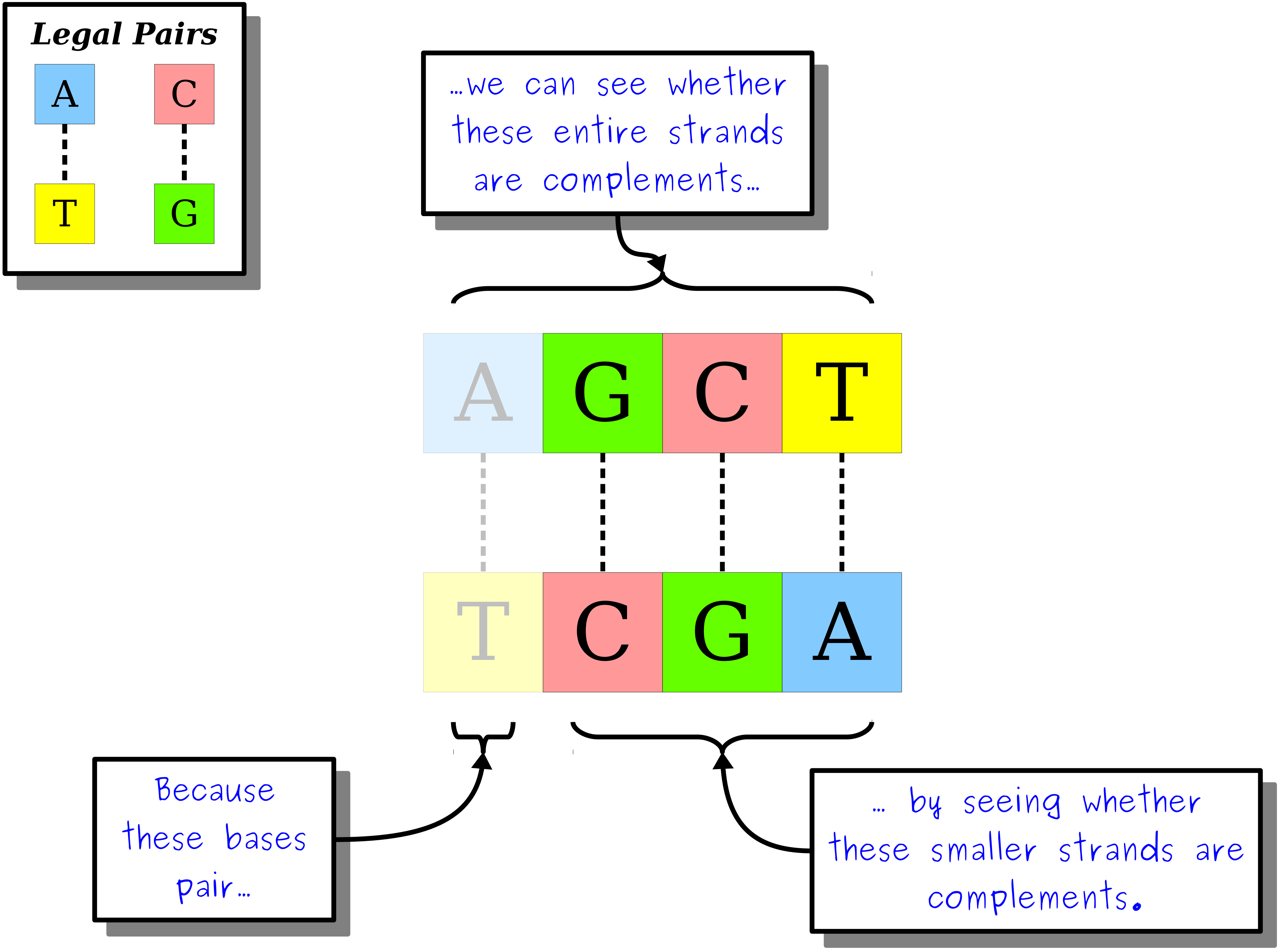
- Let’s write a recursive function

```
bool areComplementary(string one, string two);
```

that takes as input two strings representing DNA strands, then returns whether they’re complementary.

- Questions to keep in mind as we work through this:
  - What are our ***base cases***? That is, what are the simplest cases we can consider?
  - What is our ***recursive step***? That is, how do we simplify the problem down?



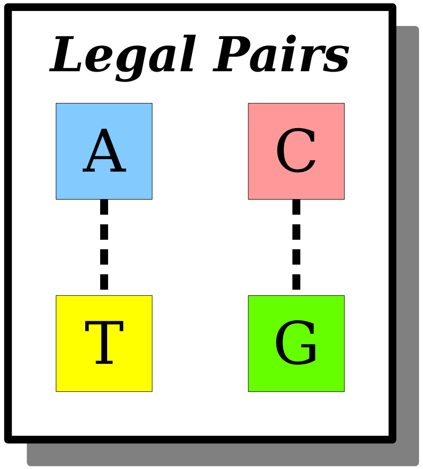


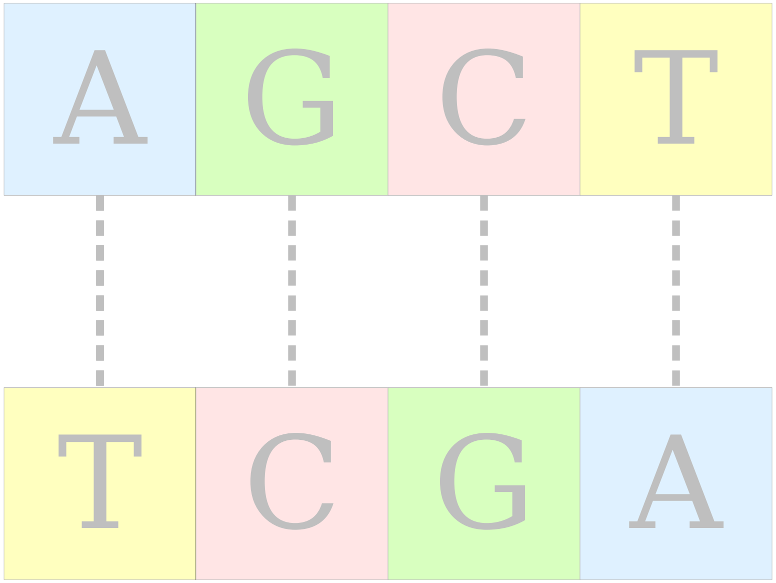
…we can see whether these entire strands are complements…

Because these bases pair…

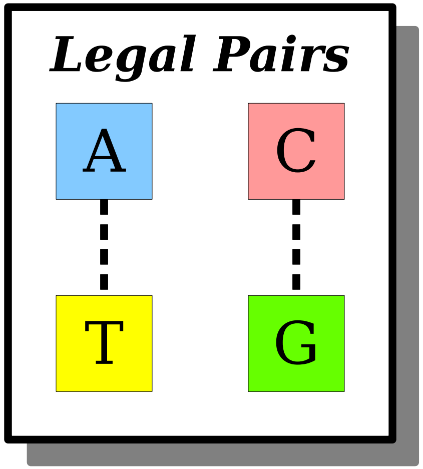
… by seeing whether these smaller strands are complements.

Legal Pairs

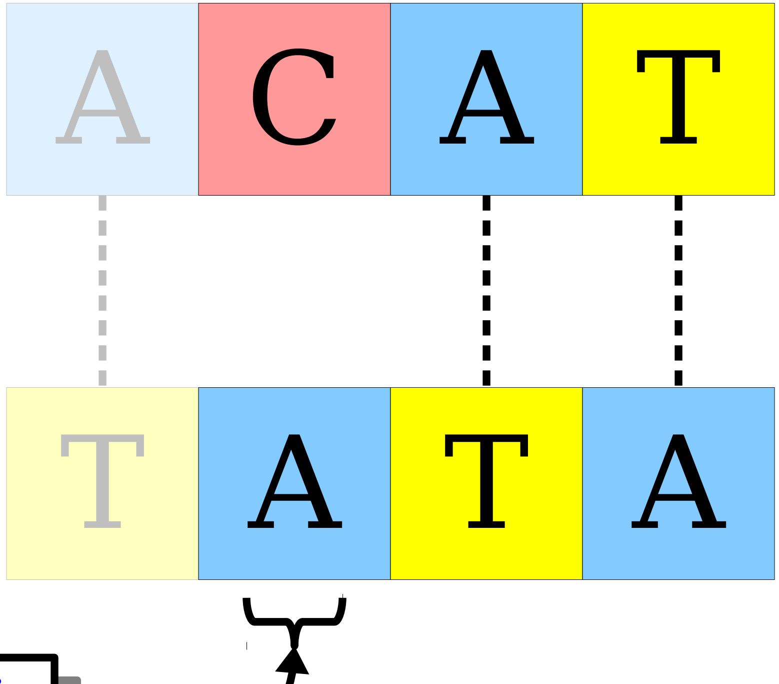




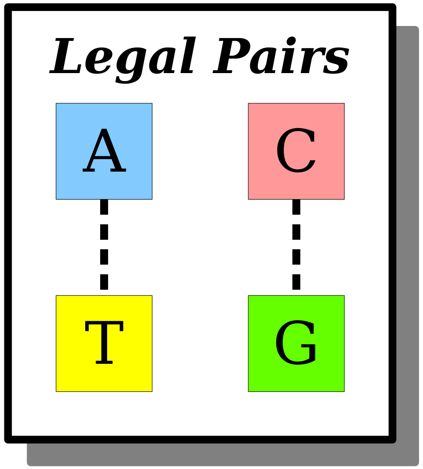
If both strands are empty, then they complement one another because there aren't any mismatches!



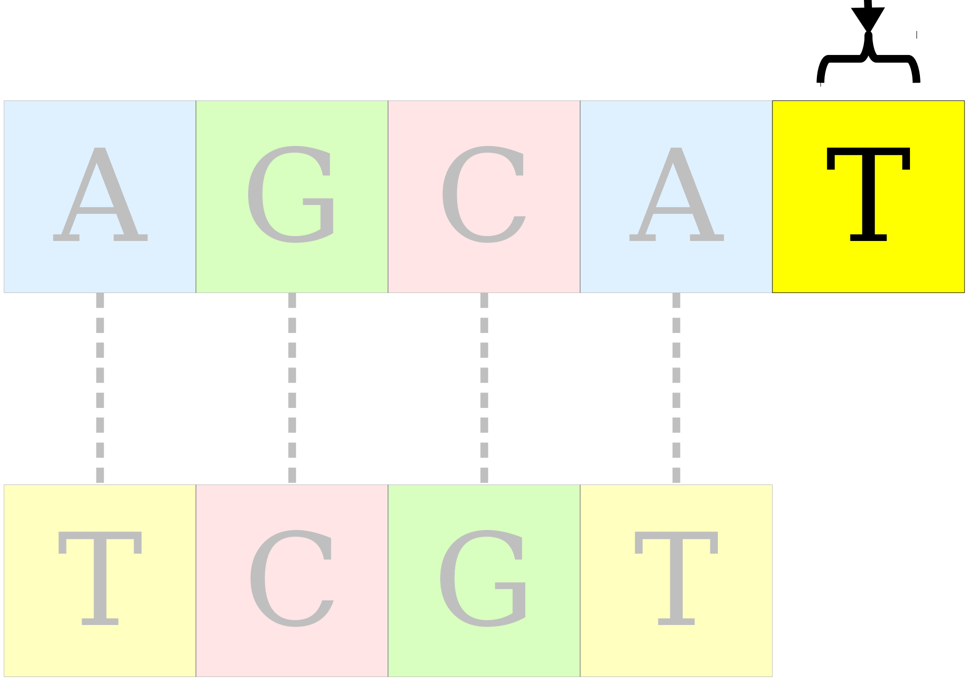
Legal Pairs



The first characters of these strands don't pair, so we're done!

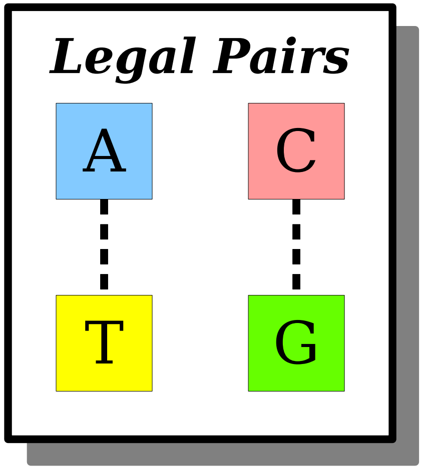


The other string is empty, so this letter can't pair with anything!



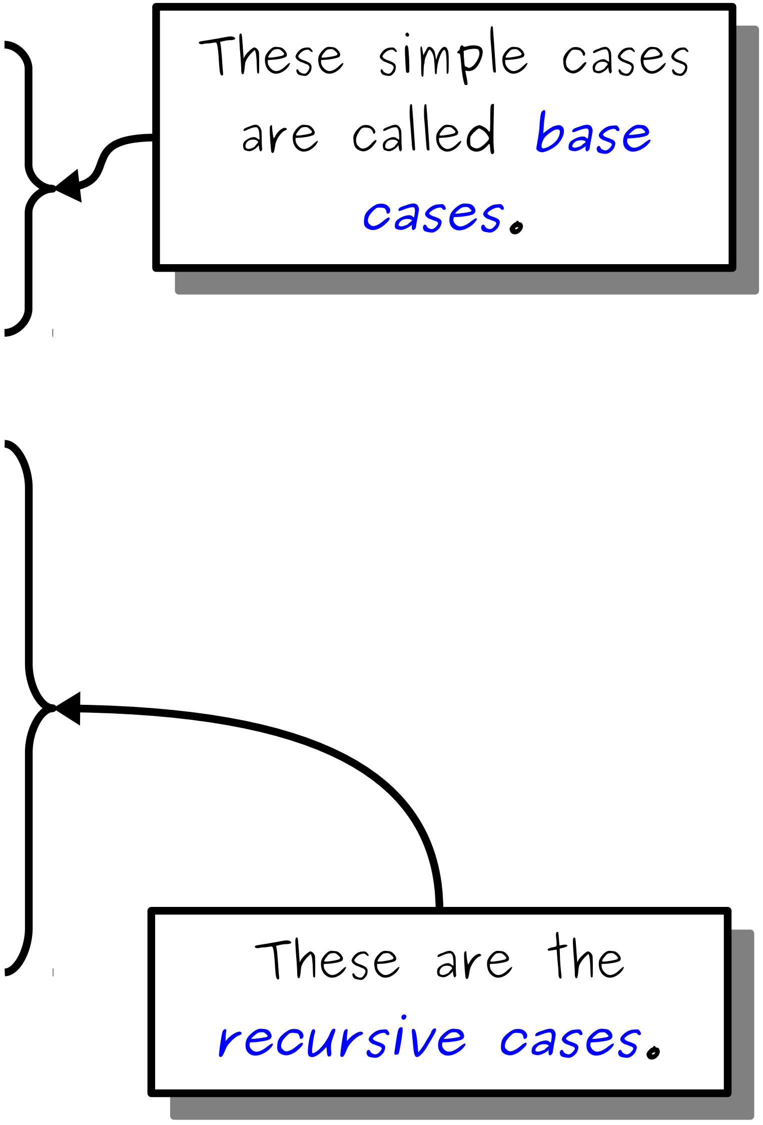
DNA Strands

- ***Base Cases:***
  - If both strands are empty, they are complementary.
  - If one strand is empty and the other isn’t, they are not complementary.
  - If the first characters don’t pair, they are not complementary.
- ***Recursive Step:***
  - If the first characters do match, drop them and see whether the rest matches.


Legal Pairs

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 and Chapter 4 of the textbook to learn more about strings and to get an intro to file processing.
- 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. ☺