# Strings and Streams

Friday Four Square!
Today at 4:15PM outside Gates.

## Announcements

- Four Handouts Today:
  - **Honor Code**
  - **Assignment 1: Welcome to C++!**
  - Submitting Assignments
  - Debugging with Visual Studio/Xcode
- Assignment 1 (Welcome to C++!) out, due Monday, April 15 at 2:15PM.
  - Warm up with C++!
  - Play around with strings and recursion!

## Announcements

- Dinner talk about patent law in technology with Twitter head of IP litigation next **Tuesday, April 9** at **7:30PM** in **Gates 219**.
- RSVP through email link sent out yesterday.

## Announcements

- Casual dinner for women studying CS next **Wednesday, April 10** at 5:00PM at the Gates Patio.
- Everyone is welcome!
- RSVP through link sent out earlier today, or by visiting

**http://bit.ly/casualcsdinner**

# 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 **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 Dawson.

## Section Signups

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

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

## Strings

## Strings

- A **string** is a (possibly empty) sequence of characters.
- Strings in C++ are conceptually similar to strings in Java.
- There are several minor differences:
    - Different names for similar methods.
    - Different behavior for similar methods
- And some really major differences:
    - Two types of 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

  ```
  str.length()
  ```
- You can read a single character in a string by writing

  ```
  str[index]
  ```
- Despite the above syntax, C++ strings are not arrays; it's just a convenient syntactic shortcut.

# Operations on Characters

- In C++, the header **`<cctype>`** contains a variety of useful functions that you can apply to characters.
- The following functions check whether a character is of a given type:

```
      isalpha  isdigit
isalnum  islower  isupper
      isspace  ispunct
```

# Strings are Mutable

- Unlike Java strings, C++ strings are mutable and can be modified.
- Change an individual character:

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

- Append more text:

```
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 search a string for some other string by using `find` (instead of `indexOf`). `find` returns `string::npos` instead of -1 if the string isn't found:

```
if (str1.find(str2) != string::npos) {
    /* found str2 inside str1 */
}
```

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

```
string allButFirstChar = str.substr(1);
string lastFiveChars = str.substr(str.length() - 5, 5);
```

## Even More Differences

- In Java, you can concatenate just about anything with a string.
- In C++, you can only concatenate strings and characters onto other strings.
- We provide a library "`strlib.h`" to make this easier.

```
string s = "I like " + integerToString(137);
```

## And the Biggest Difference

- In C++, there are two types of strings:
  - C-style strings, inherited from the C programming language, and
  - C++ `strings`, a library implemented in C++.
- Any string literal is a C-style string.
- Almost none of the operations we've just described work on C-style strings.
- Takeaway point: Be careful with string literals in C++.
  - Use the `string` type whenever possible.

```
string s = "Nubian " + "ibex";
string s = "Nubian " + "ibex";
```

Each of these strings is a C-style string, and C-style strings cannot be added with +. This code doesn't compile.

```
string s = "Nubian " + "ibex";
string s = string("Nubian ") + "ibex";
string s = string("Nubian ") + "ibex";
```

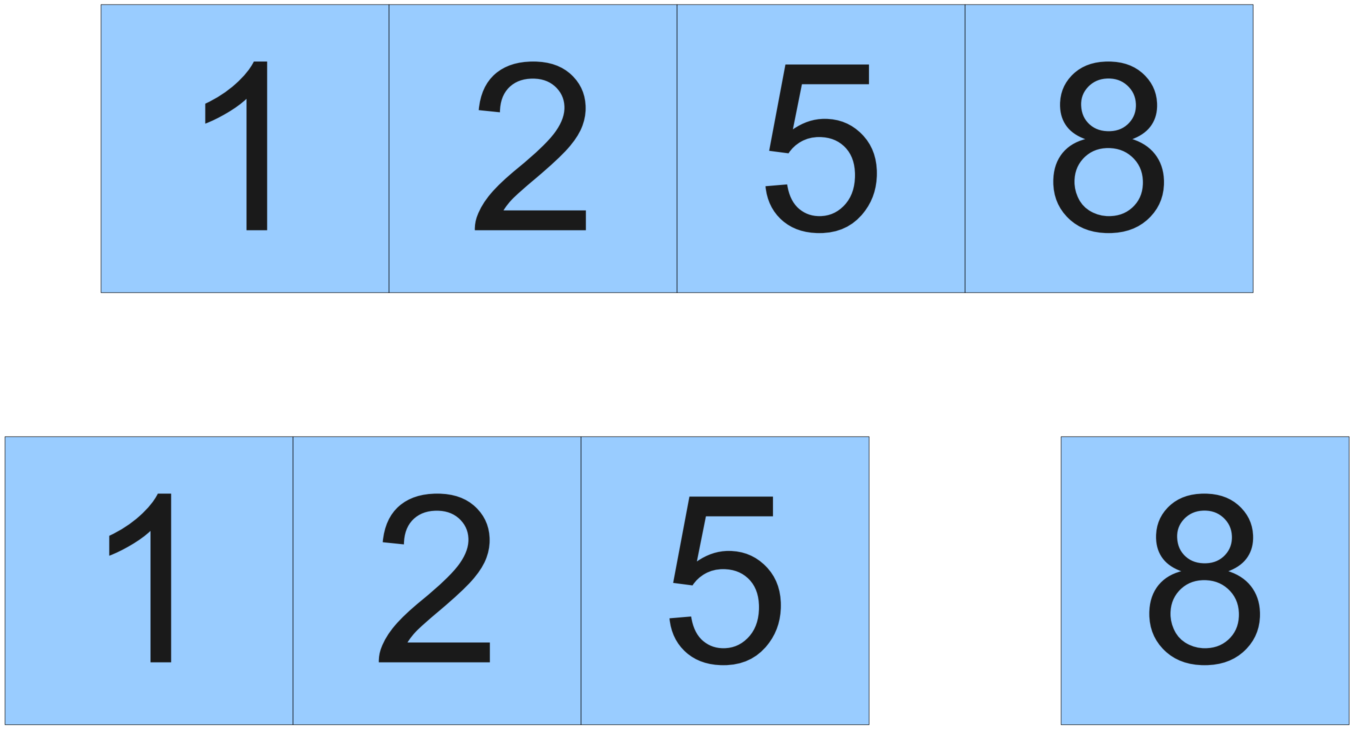
Now that we explicitly add a cast from a C-style string to a C++-style string, this code is legal. If you need to perform concatenations like this ones, make sure to cast at least one of the string literals to a C++ string.

## Recursion and Strings

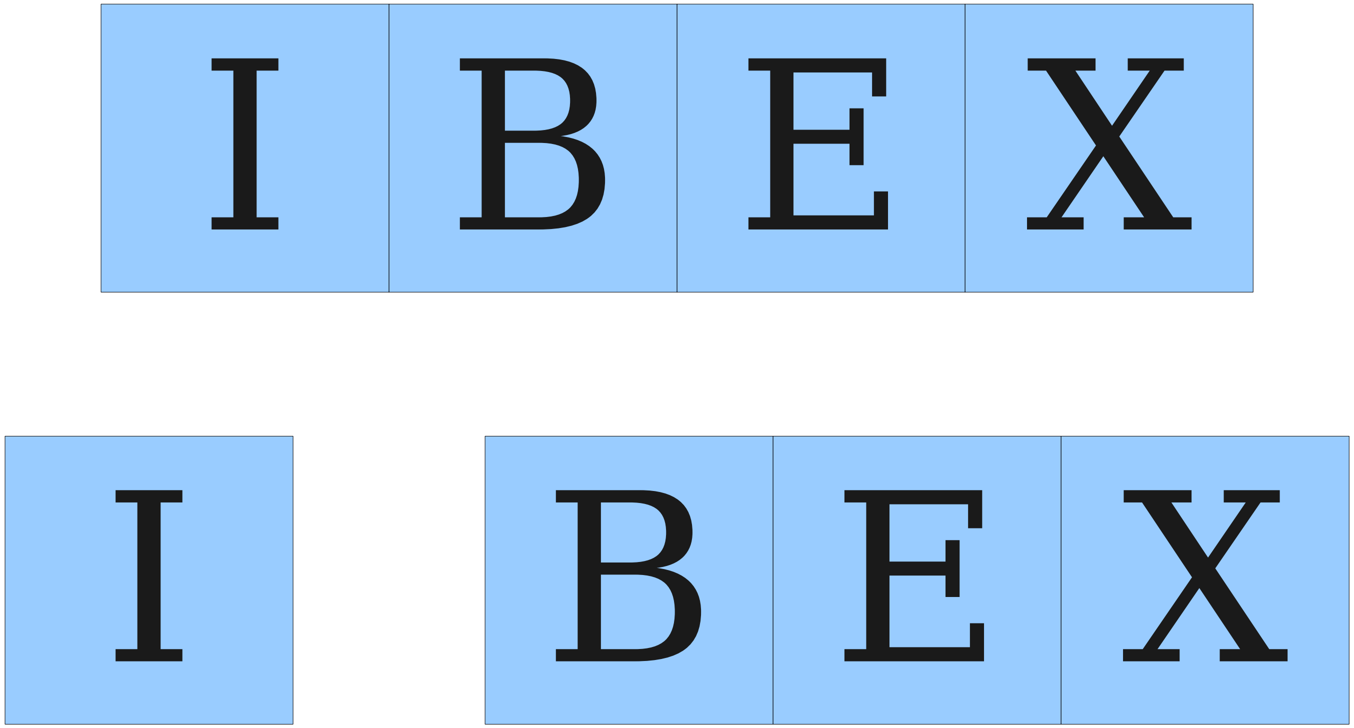
## Thinking Recursively

```
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.
}
```

# Thinking Recursively



# Thinking Recursively



## Reversing a String

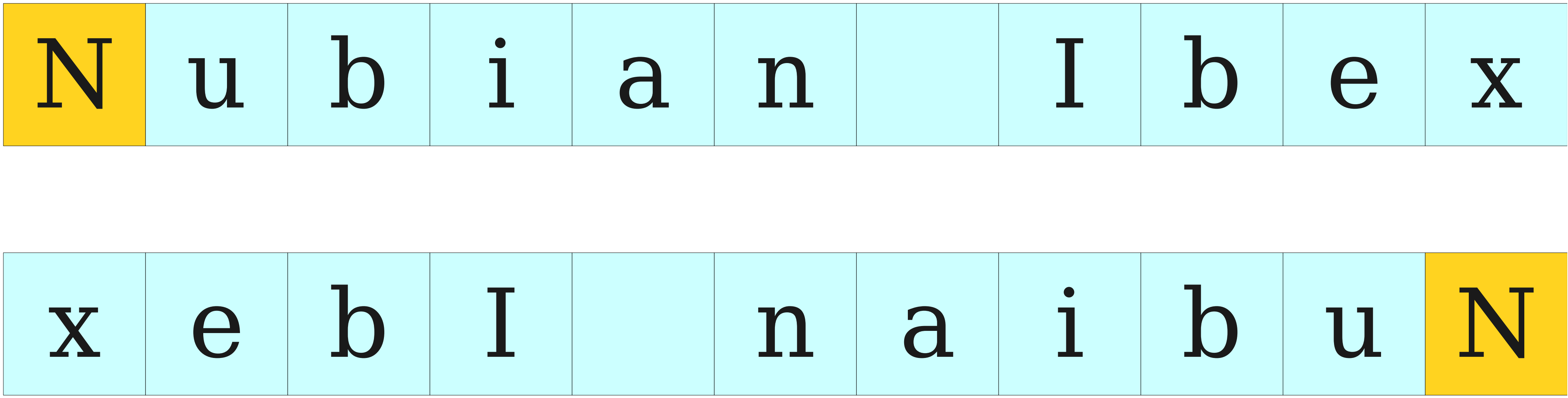
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| N | u | b | i | a | n | | I | b | e | x |

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

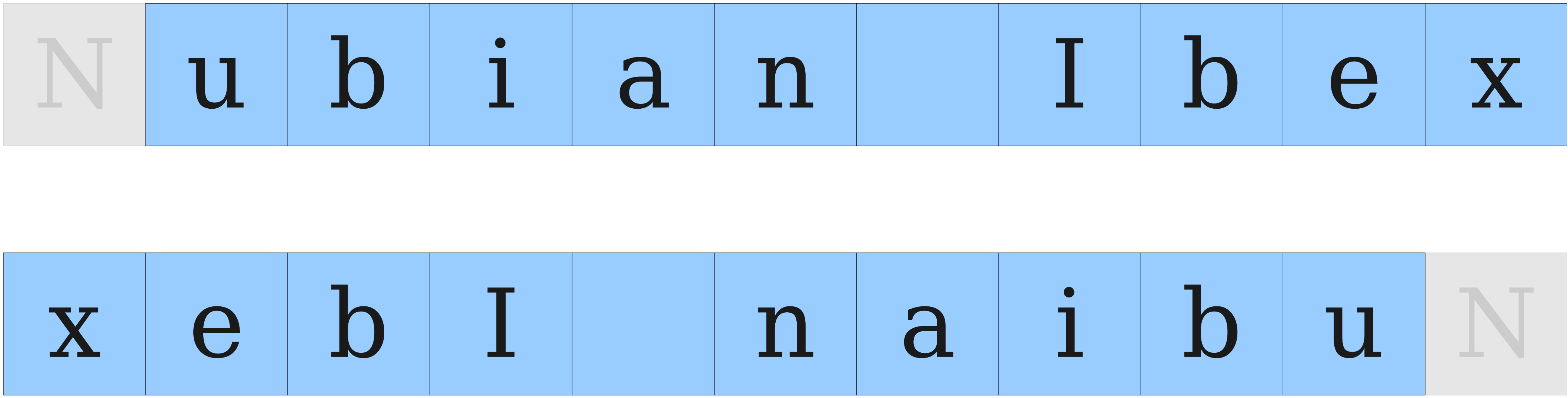
## Reversing a String

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

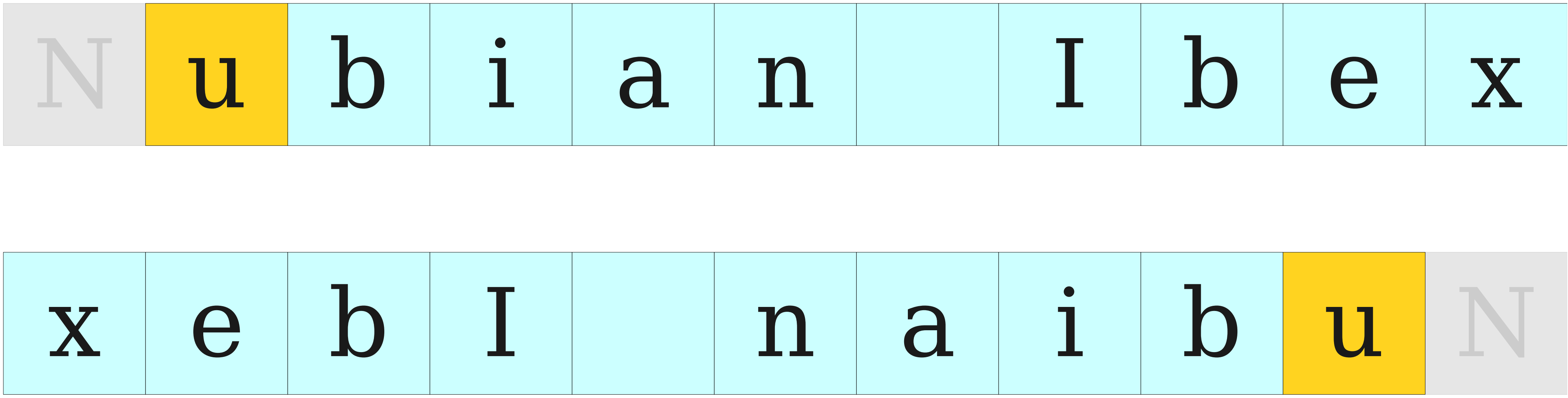
# Reversing a String



# Reversing a String



# Reversing a String



## Reversing a String Recursively

reverse("TOP")

# Reversing a String Recursively

reverse("$\boxed{T}\boxed{O}\boxed{P}$") = reverse("$\boxed{O}\boxed{P}$") + $\boxed{T}$

# Reversing a String Recursively

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

reverse("OP")
```

## Reversing a String Recursively

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

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

# Reversing a String Recursively

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

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

reverse("P")
```

# Reversing a String Recursively

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

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

reverse("P") = reverse("") + P
```

## Reversing a String Recursively

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

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

reverse("P") = reverse("") + P

reverse("") = ""
```

# Reversing a String Recursively

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

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

reverse("P") =                 "" + P

reverse("") = ""
```

# Reversing a String Recursively

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

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

reverse("P") =               P

reverse("") = ""
```

# Reversing a String Recursively

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

reverse("OP") =                P + O

reverse("P") =                 P

reverse("") = ""
```

# Reversing a String Recursively

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

reverse("OP") =                   PO

reverse("P") =                    P

reverse("") = ""
```

# Reversing a String Recursively

reverse("$\boxed{T}\boxed{O}\boxed{P}$") = $\boxed{P}\boxed{O}$ + $\boxed{T}$

reverse("$\boxed{O}\boxed{P}$") = $\boxed{P}\boxed{O}$

reverse("$\boxed{P}$") = $\boxed{P}$

reverse("") = ""

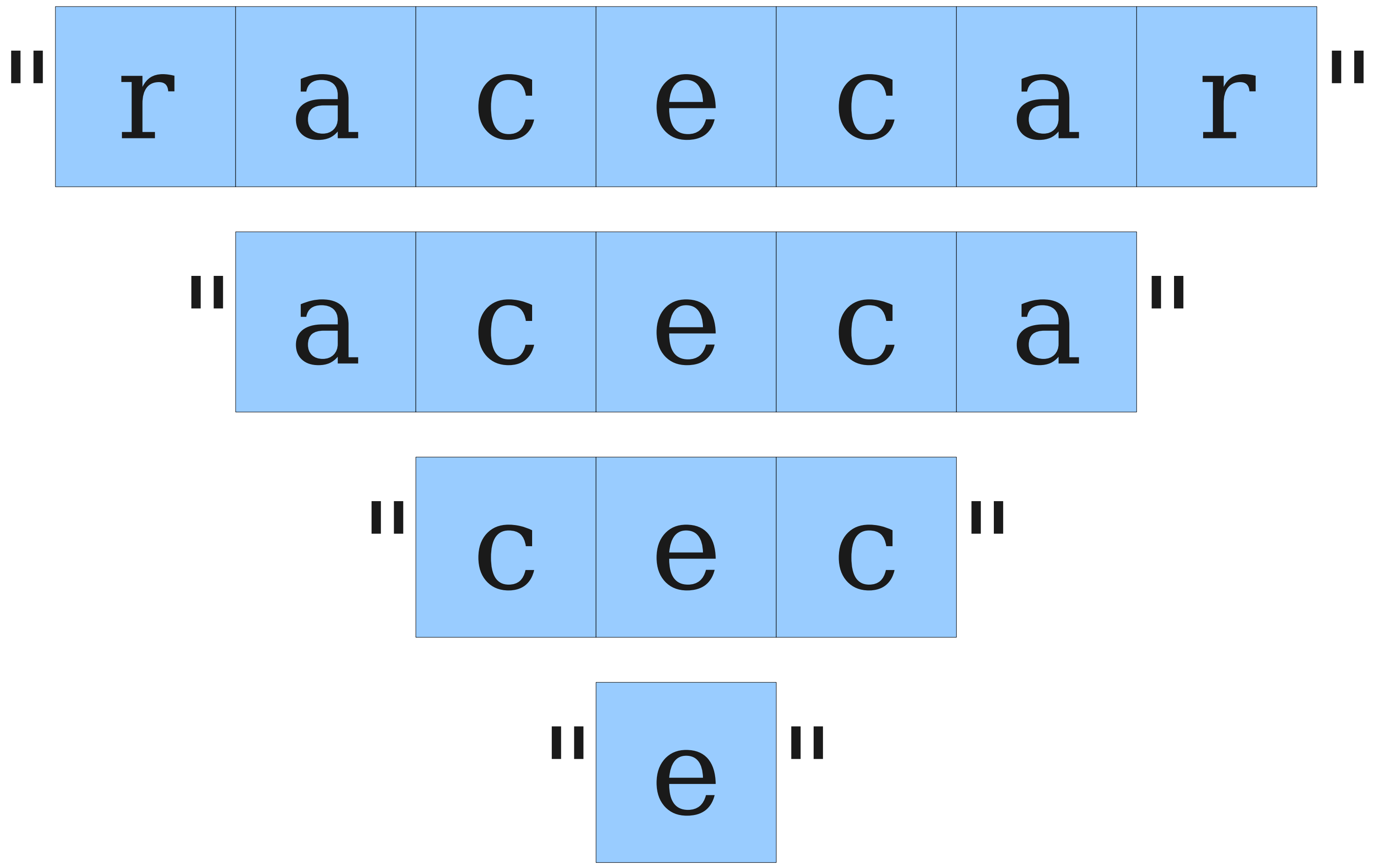
# Reversing a String Recursively

```
reverse("TOP") =          POT

reverse("OP") =           PO

reverse("P") =            P

reverse("") = ""
```

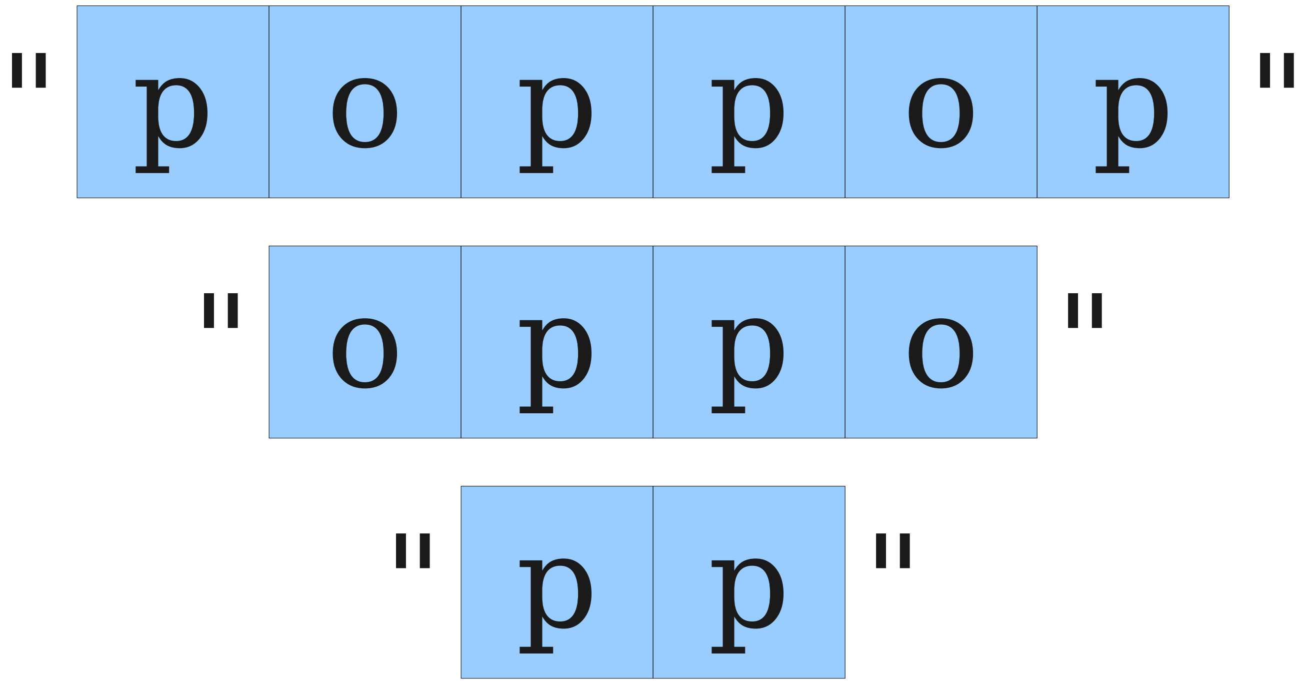
## Palindromes

- A palindrome is a string whose letters are the same forwards and backwards.
- For example:
  - Go hang a salami!  I'm a lasagna hog.
  - Mr. Owl ate my metal worm.
  - Anne, I vote more cars race Rome to Vienna.

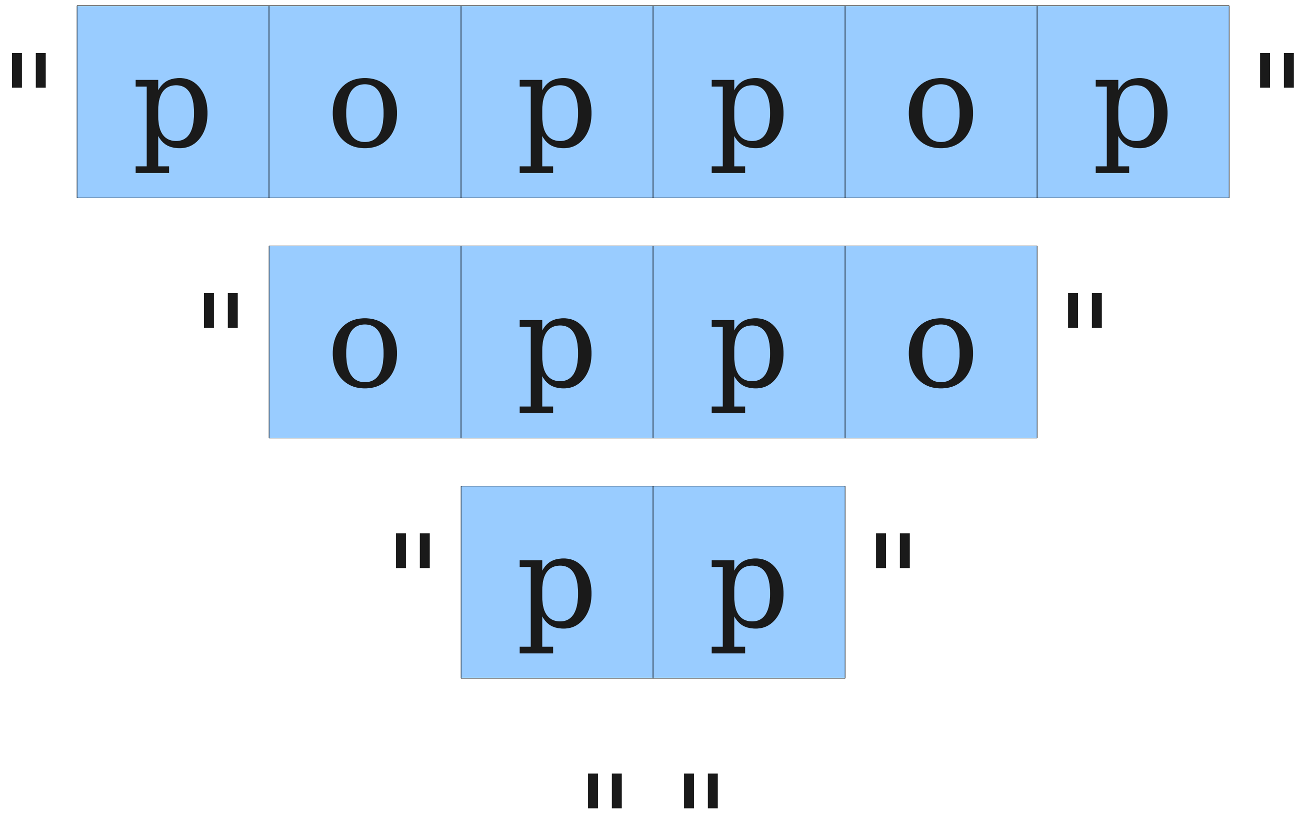
# Thinking Recursively



# Thinking Recursively



# Thinking Recursively



## Getting Data from Files

- Now that we have collections classes, we can start working with data pulled in from external files.
- File reading in C++ is done using the **`ifstream`** class.
  - Must **`#include`** `<fstream>` to use `ifstream`.

# Reading Line by Line

- You can read a line out of an `ifstream` by using the **`getline`** function:

  `getline(`***file***`, `***str***`)`

- The canonical "read each line of a file loop" is shown here:

```
string line;
while (getline(file, line)) {
    /* … process line … */
}
```

- Chapter 4 of the course reader has more details about file I/O in C++; highly recommended!

# Reading Formatted Data

- You can read formatted data from a file by using the **stream extraction operator**:

  ***file*** >> ***variable***

- Can read any primitive type, plus strings.
- When reading strings, stops at newlines or whitespace.
- Canonical "read formatted data loop:"

```
type val;
while (file >> val) {
    /* … process val … */
}
```

# Next Time

- **`Stack`**
  - A surprisingly useful collection class.
- **`TokenScanner`**
  - A tool for cutting apart strings.
- **The Shunting-Yard Algorithm**
  - How do computers parse expressions?