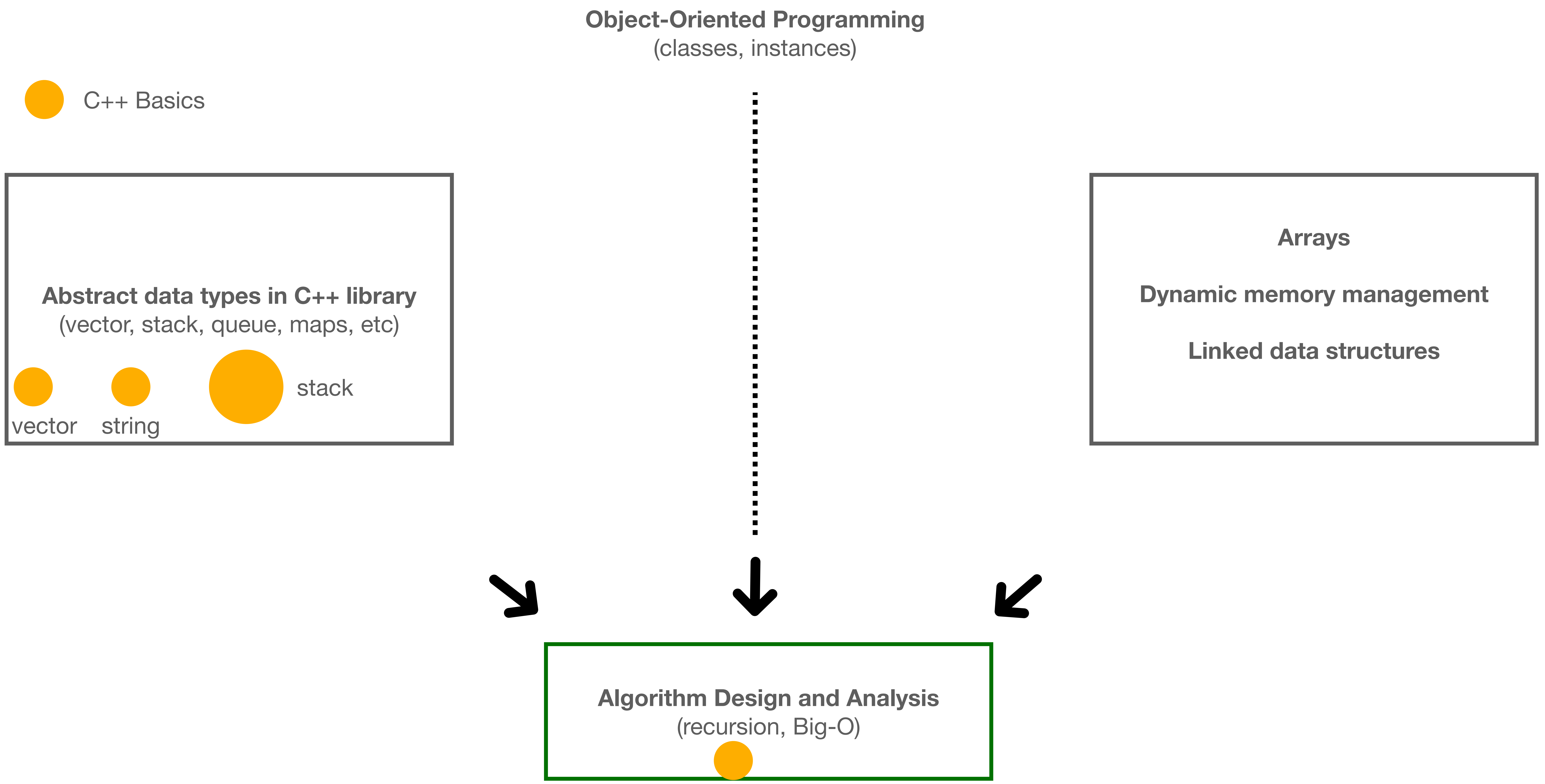
Lecture5: Stack

Xingyu Zhou

09/15 2025

* Part of the materials are inspired or borrowed from the course CS 106B at Stanford

Roadmap


Object-Oriented Programming
(classes, instances)
C++ Basics
Abstract data types in C++ library
(vector, stack, queue, maps, etc)
vector
string
stack
Arrays
Dynamic memory management
Linked data structures
Algorithm Design and Analysis
(recursion, Big-O)

Outline of Today's Class

- Review
- Stack
- Application: Balanced Parenthesis
- In-class problem

Review…

std::string

#include <string>

h/cpp hackingcpp.com

```
string s = "I'm sorry, Dave.";
            0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15   indices
```

non-mutating

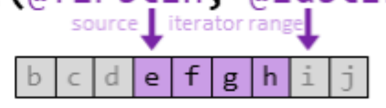
`s.size()`	→	`16`	(number of characters)
`s[2]`	→	`'m'`	(character at index 2)
`s.find("r")`	→	`6`	(first match from start)
`s.rfind("r")`	→	`7`	(first match from end)
`s.find("X")`	→	`string::npos`	(not found, invalid index)
`s.find(' ', 5)`	→	`10`	(first match after index ≥ 5)
`s.substr(4, 6)`	→	`string{"sorry,"}`	
`s.contains("sorry")`	→	*true*	(C++23)
`s.starts_with('I')`	→	*true*	(C++20)
`s.ends_with("Dave.")`	→	*true*	(C++20)
`s.compare("I'm sorry, Dave.")`	→	`0`	(identical)
`s.compare("I'm sorry, Anna.")`	→	`> 0`	(same length, but 'D' > 'A')
`s.compare("I'm sorry, Saul.")`	→	`< 0`	(same length, but 'D' < 'S')

mutating

	`s += " I'm afraid I can't do that."`	⇒	`s = "I'm sorry, Dave. I'm afraid I can't do that."`
	`s.append("..")`	⇒	`s = "I'm sorry, Dave..."`
size	`s.clear()`	⇒	`s = ""`
size	`s.resize(3)`	⇒	`s = "I'm"`
size	`s.resize(20, '?')`	⇒	`s = "I'm sorry, Dave.????";`
index based	`s.insert(4, "very ")`	⇒	`s = "I'm very sorry, Dave."`
index based	`s.erase(5, 2)`	⇒	`s = "I'm sry, Dave."`
index based	`s[15] = '!'`	⇒	`s = "I'm sorry, Dave!"`
index based	`s.replace(11, 5, "Frank")`	⇒	`s = "I'm sorry, Frank"`
iterator based	`s.insert(s.begin(), "HAL: ")`	⇒	`s = "HAL: I'm sorry, Dave."`
iterator based	`s.insert(s.begin()+4, "very ")`	⇒	`s = "I'm very sorry, Dave."`
iterator based	`s.erase(s.begin()+5)`	⇒	`s = "I'm srry, Dave."`
iterator based	`s.erase(s.begin(), s.begin()+4)`	⇒	`s = "sorry, Dave."`

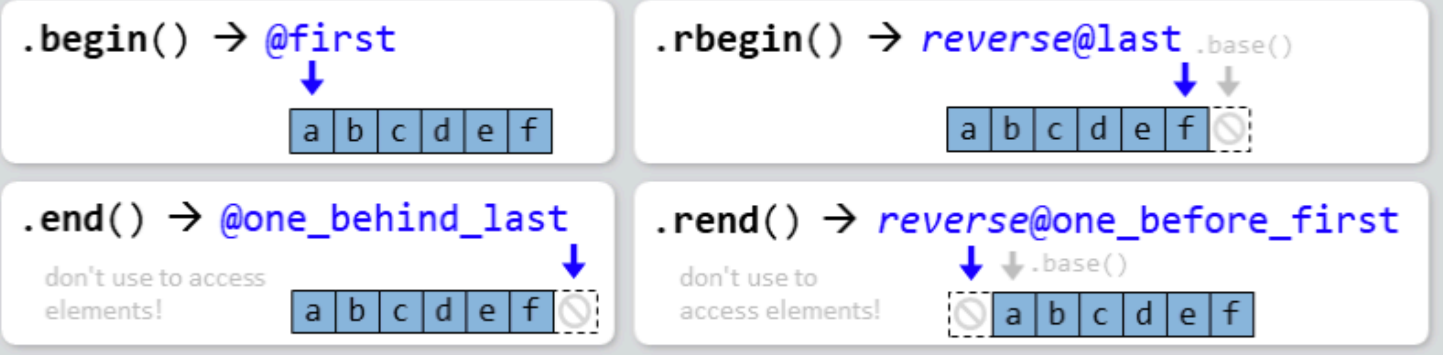
Constructors

`string{'a','b','c'}`	➡	a b c
`string(4, '$')`	➡	$ $ $ $
`string(@firstIn, @lastIn)`	➡	e f g h
`string( a b c d )` copy/move	➡	a b c d



source string object

Obtain Iterators or Reverse Iterators



.begin() → @first
a b c d e f
.end() → @one_behind_last
don't use to access elements!
.rbegin() → reverse@last
.base()
.rend() → reverse@one_before_first
don't use to access elements!
.base()

String → Number Conversion

```
              int        stoi (●,●,●);     const string&
             long        stol (●,●,●);       input string
        long long        stoll(●,●,●);
                                           std::size_t* p = nullptr
    unsigned long        stoul (●,●,●);      output for
unsigned long long       stoull(●,●,●);      number of processed characters
                                           int base = 10
            float        stof (●,●,●);       base of target system;
           double        stod (●,●,●);       default: decimal
      long double        stold(●,●,●);
```

Number → String Conversion

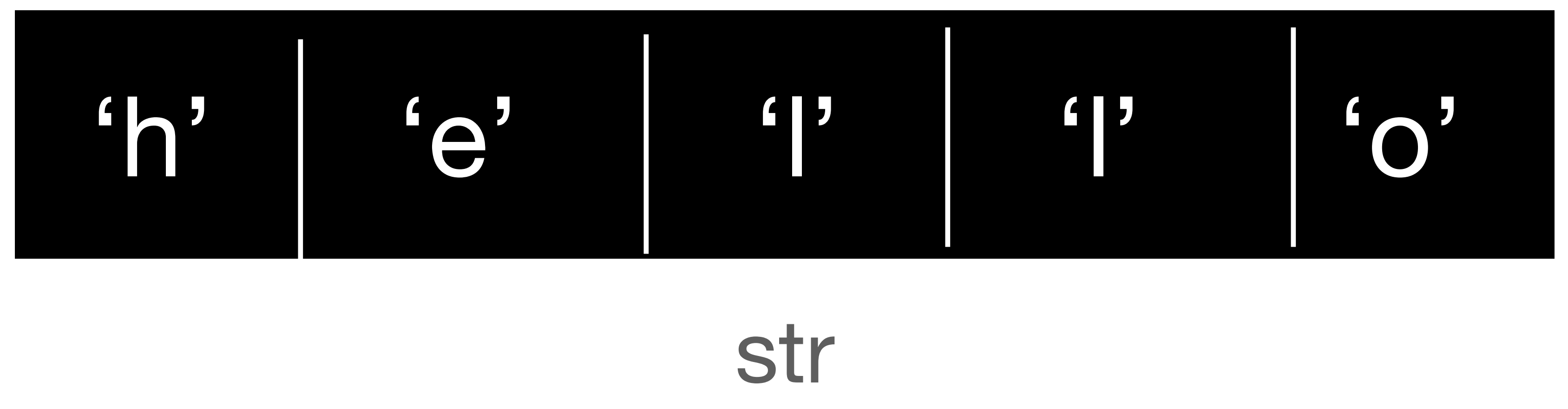
```
string to_string( ● );

  int | long | long long |
  unsigned | unsinged long | unsigned long long |
  float | double | long double
```

v1.6

C++ string

`#include<string>`

- C++ string is a sequence container of characters
  - It is similar to `vector<char>`
- `string str = “hello”;`



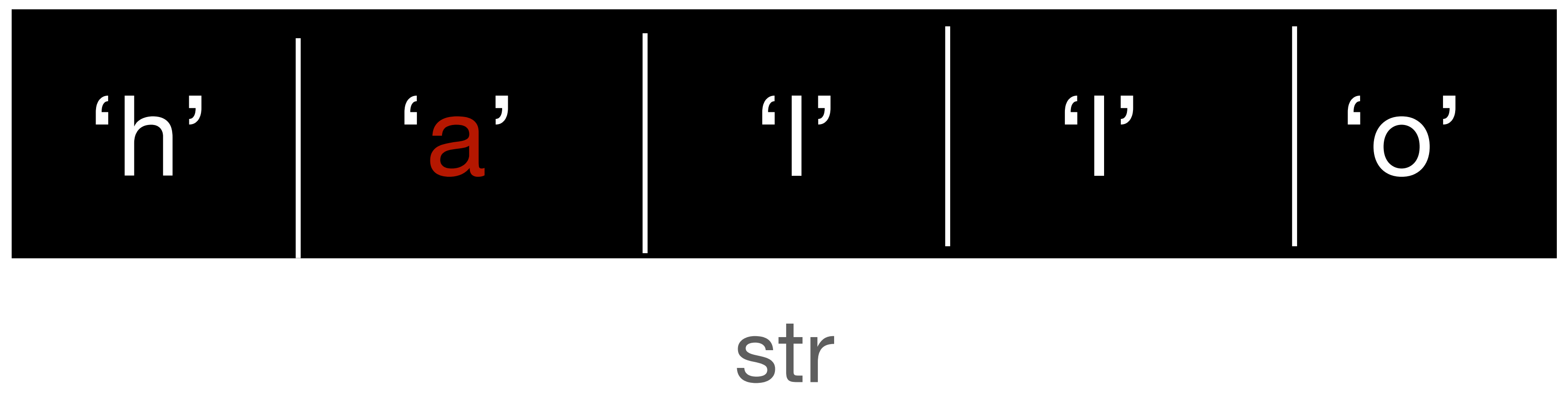
Key features of C++ string

Mutable

- C++ string is mutable
    - Unlike Python and Java

```
string str = “hello”;

str[1] = ‘a’
```

Note that we need to use single quote

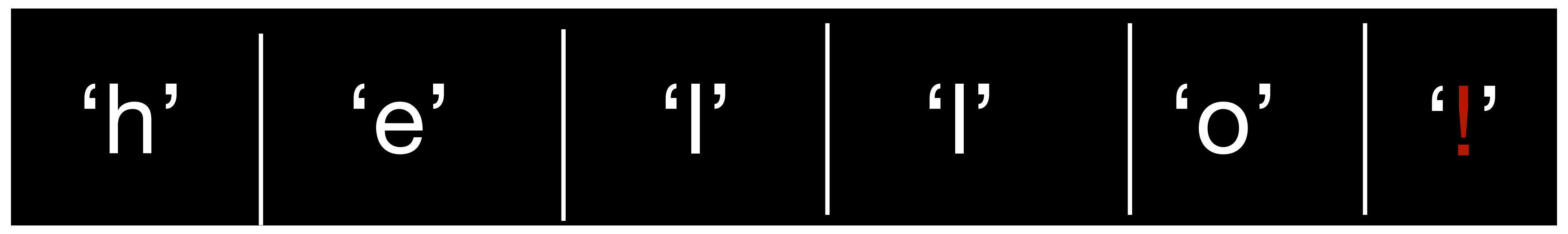


Key features of C++ string

Concatenation

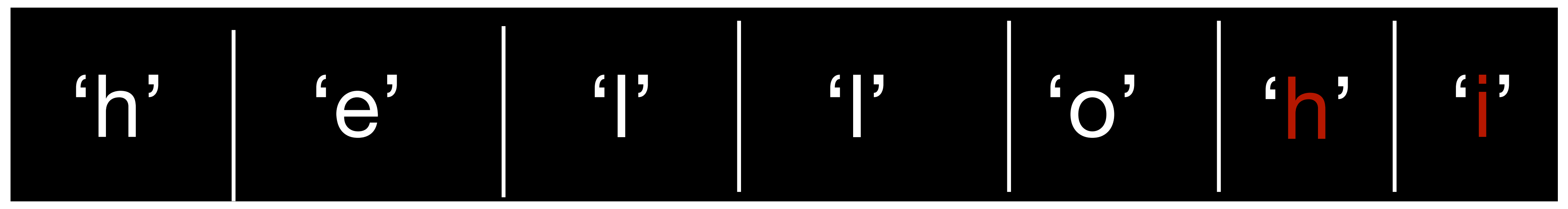
- You can add character to strings and string to strings using += and +

```
string str = “hello”;

str += ‘!’
```



str

```
string str = “hello”;

str += “hi”
```



str

Key features of C++ string

Compare

- You can use logical operators to compare strings (and characters)
  - `> = <`
  - It will use the corresponding ascii value of char

```
string str1 = “ab”;

string str2 = “ba”;

str1 < str2; // true
```

```
string str1 = “aa”;

string str2 = “AA”;

Str1 > str2; // true
```

Key features of C++ string

Loop over a string

- You can use index or directly use for each
  - For example, `string str = “hello”;`

```
for(int index = 0; index < str.length(); index++) {
   cout << str[index];
}
for(char a: str) {
   cout << a;
}
```

String utility functions, return the length of the string

C++ string utility functions

- **`s.append(str)`**: add text **`str`** to the end of a string
- **`s.compare(str)`**: return -1, 0, or 1 depending on relative ordering
- **`s.erase(index, length)`**: delete text from a string starting at given **`index`**
- **`s.find(str)`**: return first **`index`** where the start of **`str`** appears in this string (returns string::npos if not found)
- **`s.rfind(str)`**: return last **`index`** where the start of **`str`** appears in this string (returns string::npos if not found)
- **`s.insert(index, str)`**: add text **`str`** into a string at a given **`index`**
- **`s.length()`** or **`s.size()`**: number of characters in this string
- **`s.replace(index, len, str)`**: replaces **`len`** chars at **`index`** with text **`str`**
- **`s.substr(start, length)`** or **`s.substr(start)`**: the next **`length`** characters beginning at **`start`** (inclusive); if **`length`** omitted, grabs till end of string

Stack

[An ADT that follows Last-In-First-Out principle]

What is a stack?

LIFO

- An ADT represents a stack of things
- Element can only be pushed on top of the stack
- Element can only be popped from the top of the stack
  - Hence, it follows the Last-In-First-Out
- One cannot has a random access to any particular element
  - This restricts the behavior, but it also leads to a simple interface
- Stack has a wide range of applications

Stack in C++

`#include<stack>`

- `push(value)` — place an entity onto the top of the stack
- `pop()` — remove an entity from the top of the stack
- `top()` — get the entity at the top of the stack, but don’t remove it
- `empty()` — `true` if the stack is empty
- `size()` — return the number of elements in the stack

With these simple operations, one can use stack to solve many interesting real-world problems!

Live Demo

[Basics of stack in C++: https://onlinegdb.com/v50vUqnWA]

Application

[Balanced parenthesis]

Balanced Parenthesis

Given a string, determine whether the parenthesis is correct

```
void fun(){if (x[0] > 3) {y = 1};}
```

As a human, we can easily see this expression is correct in terms of parenthesis

But, how can we write a program to automatically do this for us?

Key observation: any closed parenthesis need to match the most recent open parenthesis

Remind you about something? The most recent one is the top one on the stack, right?

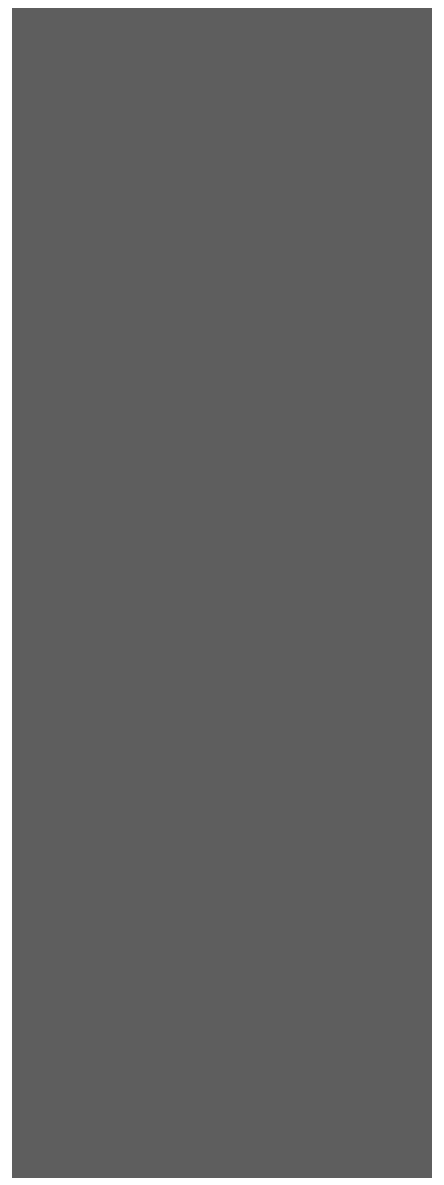
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};;}
^
```

For each char in the string:

    If it not a parenthesis, skip it…(do nothing)


Stack

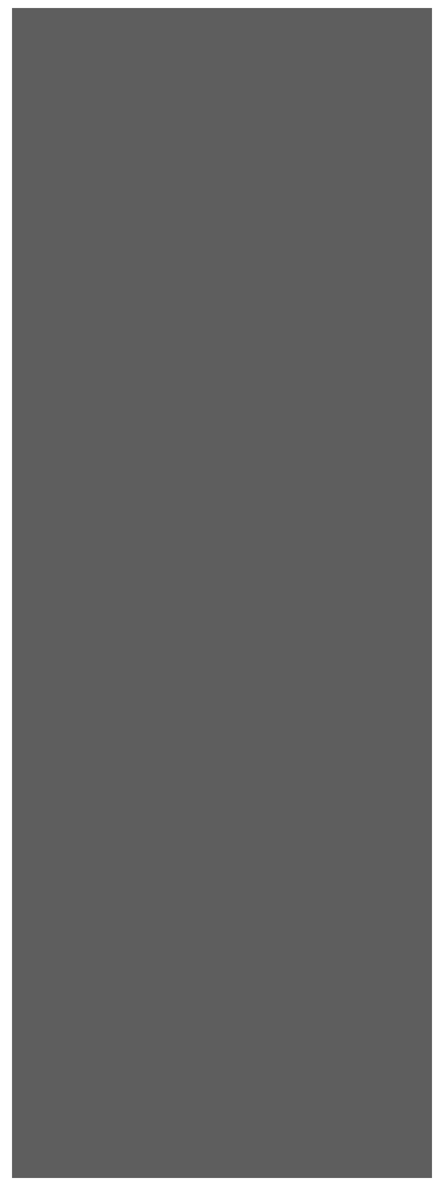
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
 ^
```

For each char in the string:

    If it not a parenthesis, skip it…(do nothing)



Stack

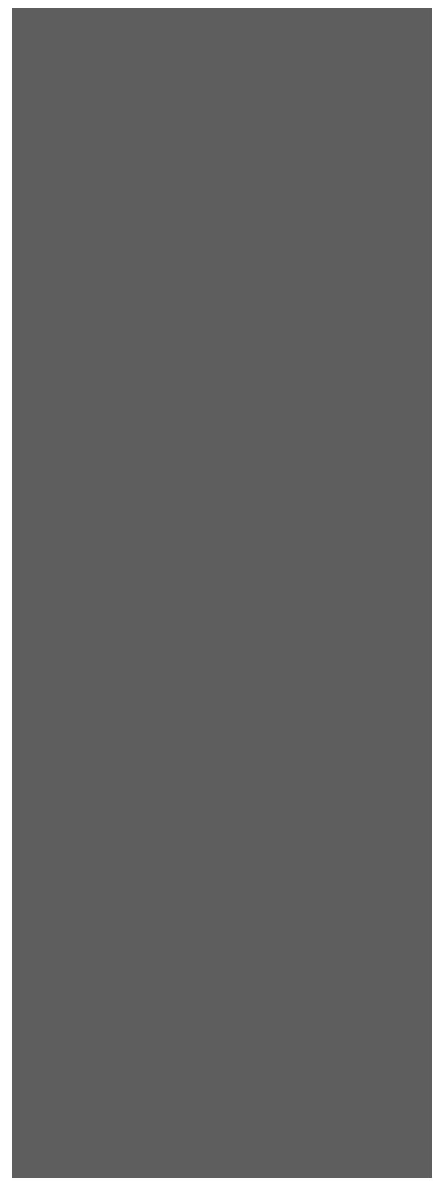
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
  ^
```

For each char in the string:

    If it not a parenthesis, skip it…(do nothing)



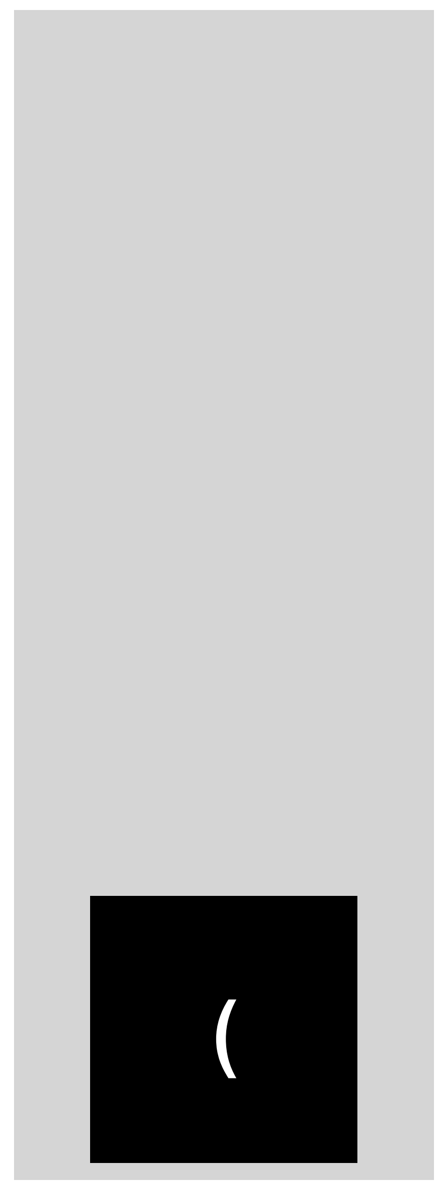
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};;}
        ^
```

For each char in the string:

  If it is a **left (open) parenthesis,** push it on the stack



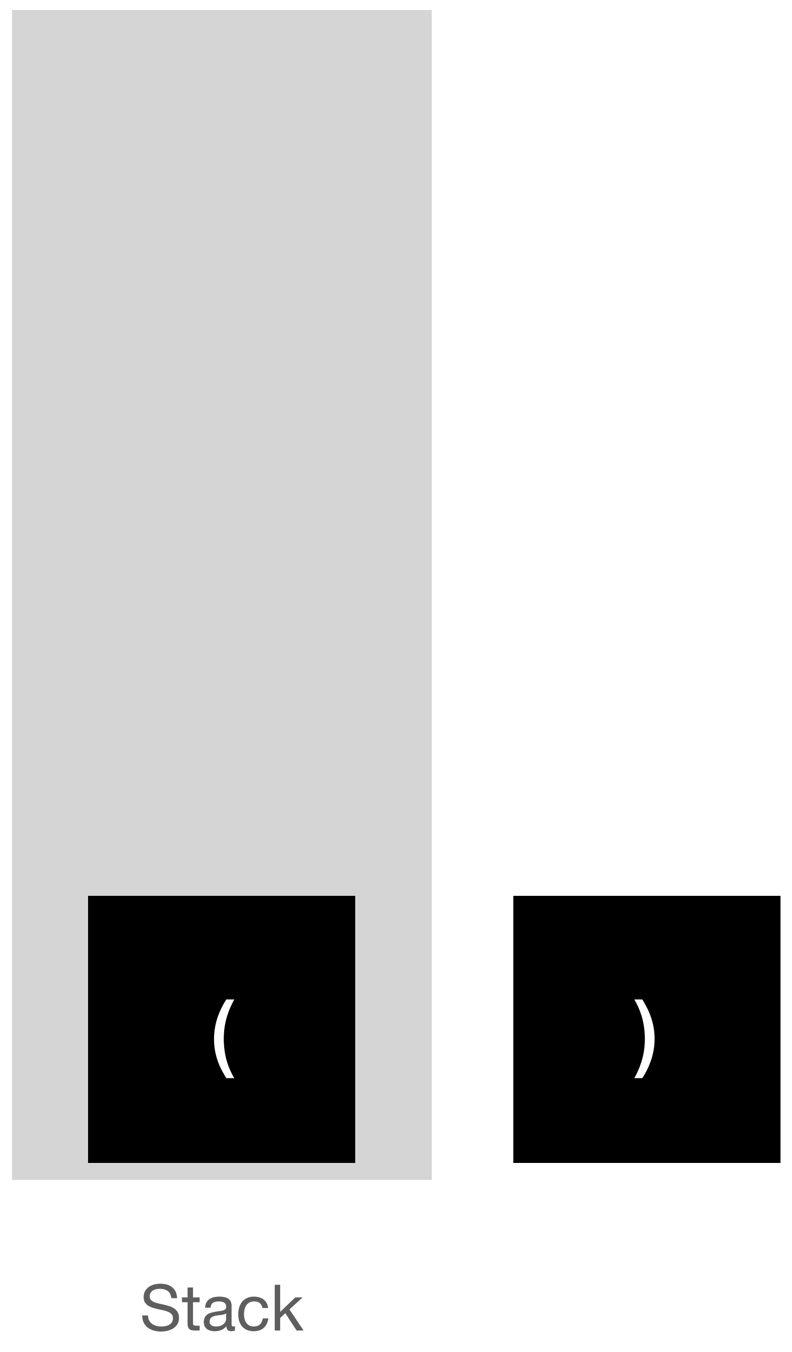
Stack

Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};;}
           ^
```

```
For each char in the string:
    If it is a **right (closed) parenthesis**
        compare it with the top (and pop it)
            If they are not matched, return false
```



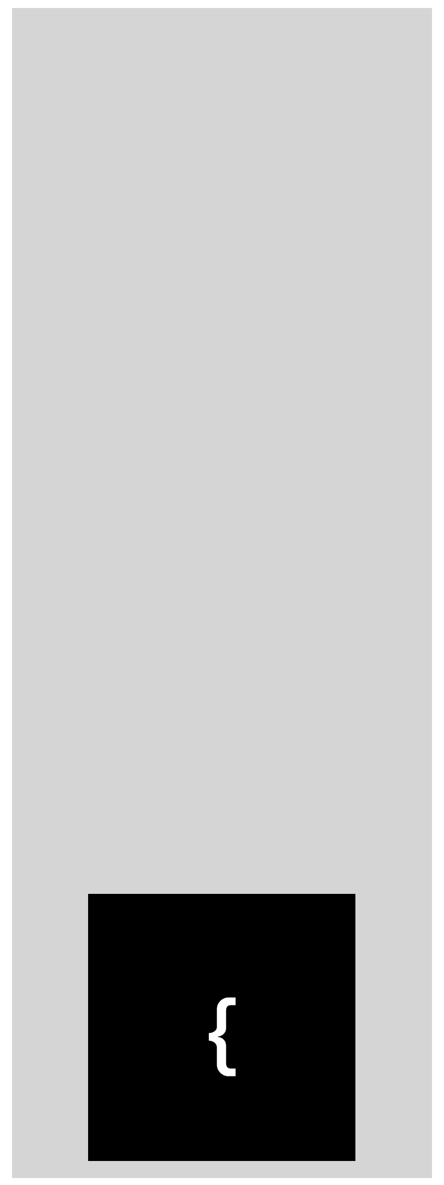
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
           ^
```

For each char in the string:

    If it is a **left (open) parenthesis,** push it on the stack



Stack

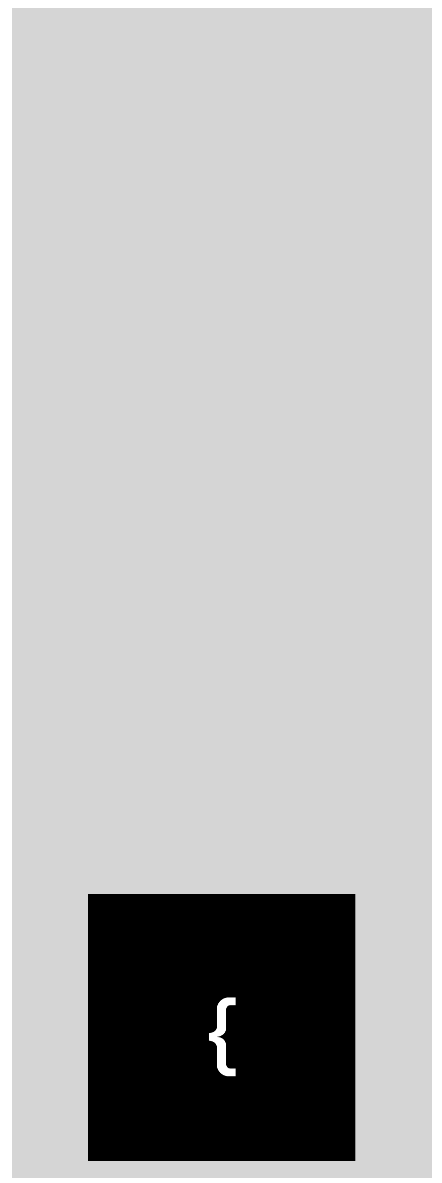
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
           ^
```

For each char in the string:

  If it not a parenthesis, skip it…



Stack

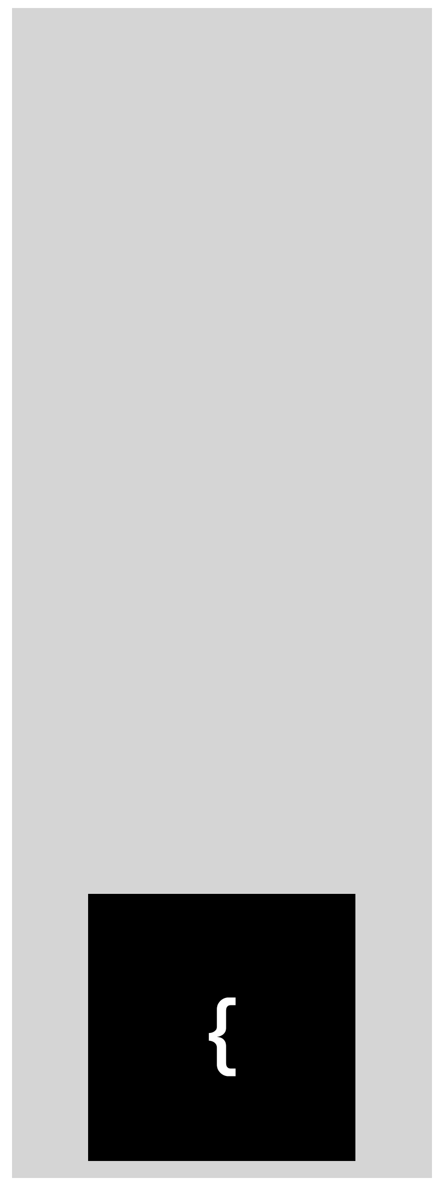
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
            ^
```

For each char in the string:

If it not a parenthesis, skip it…



Stack

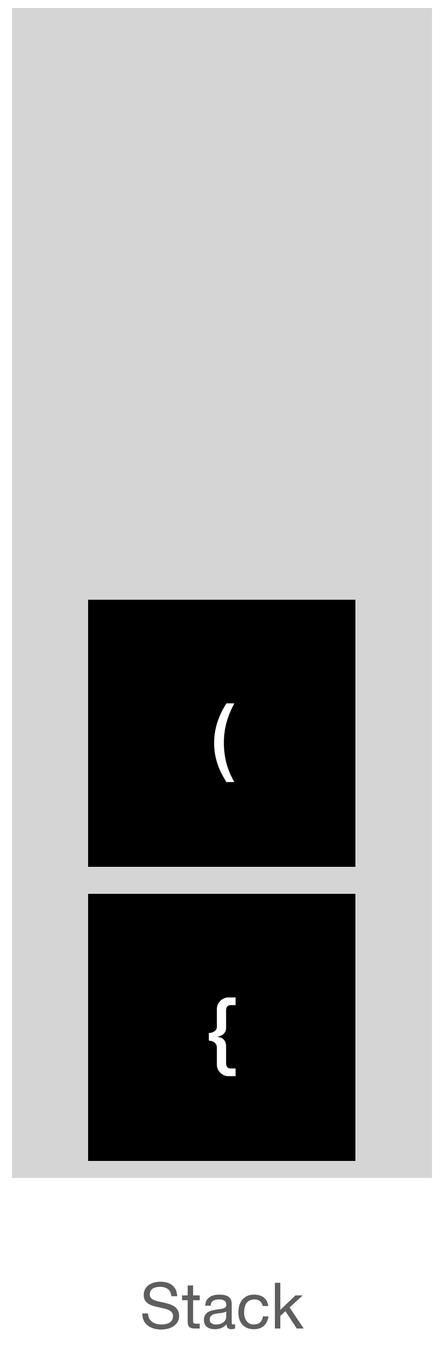
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};;}
              ^
```

For each char in the string:

If it is a **left (open) parenthesis,** push it on the stack



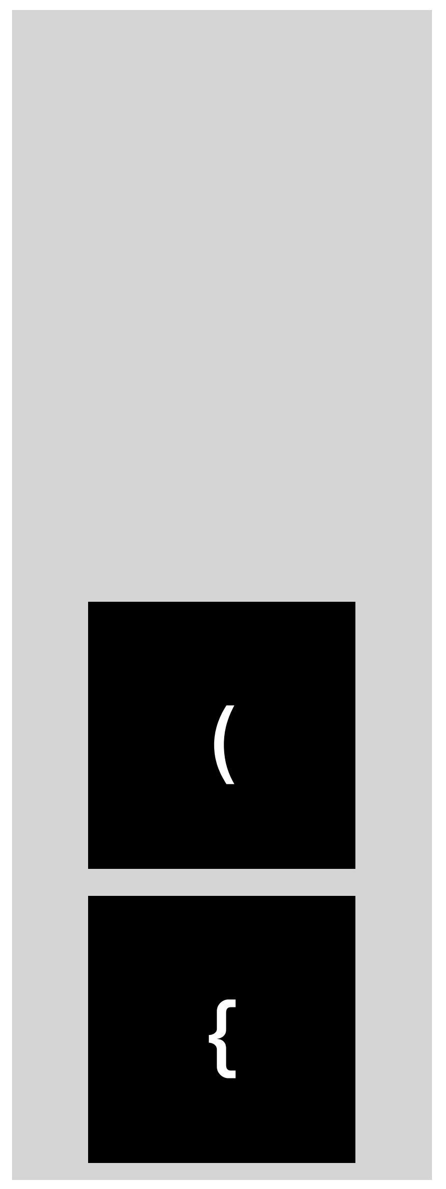
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
               ^
```

For each char in the string:

  If it not a parenthesis, skip it…



Stack

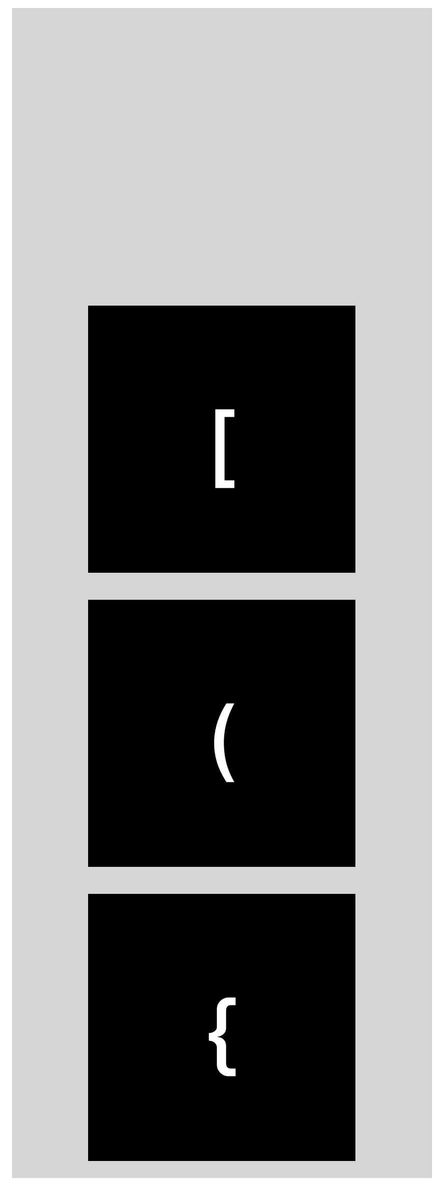
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};;}
                 ^
```

For each char in the string:

    If it is a **left (open) parenthesis,** push it on the stack



Stack

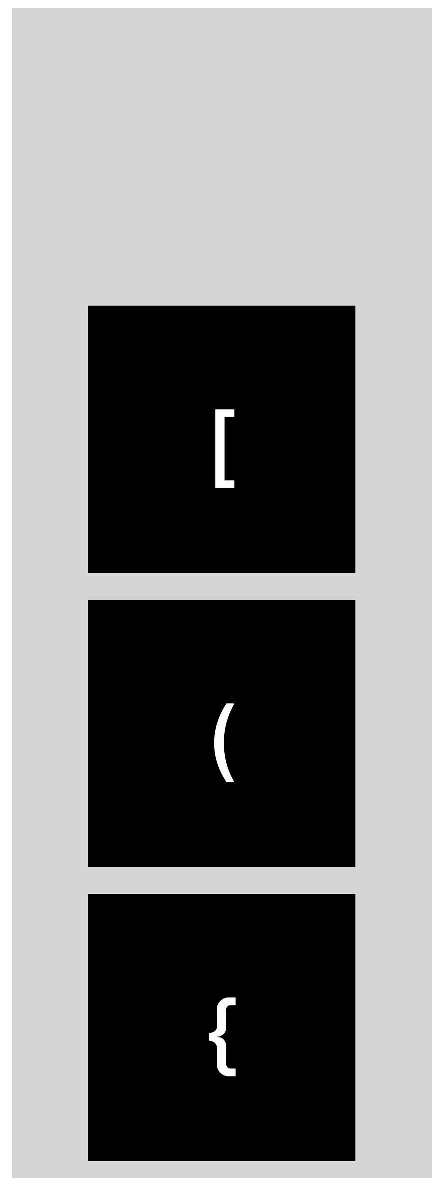
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
                ^
```

For each char in the string:

  If it not a parenthesis, skip it…



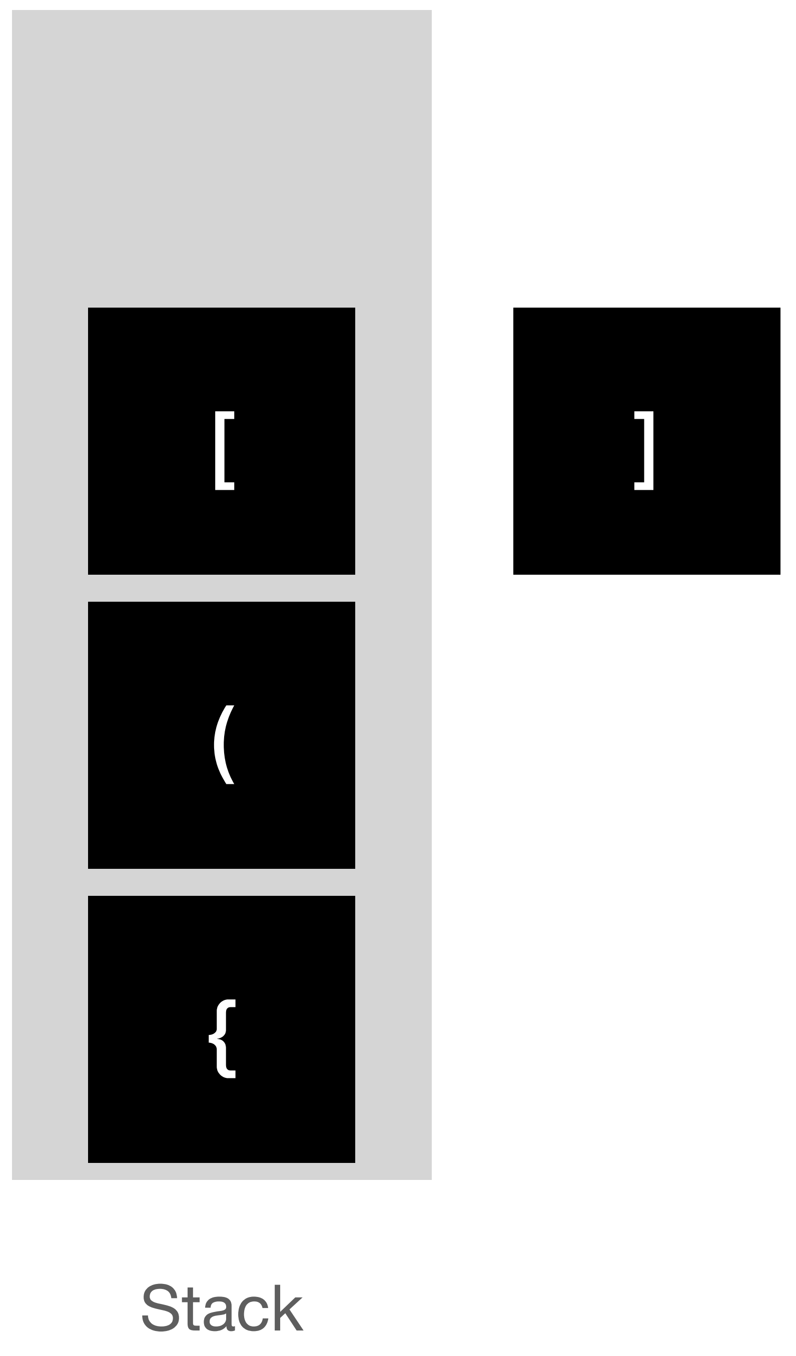
Stack

Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
                   ^
```

```
For each char in the string:
    If it is a right (closed) parenthesis
        compare it with the top (and pop it)
            If they are not matched, return false
```



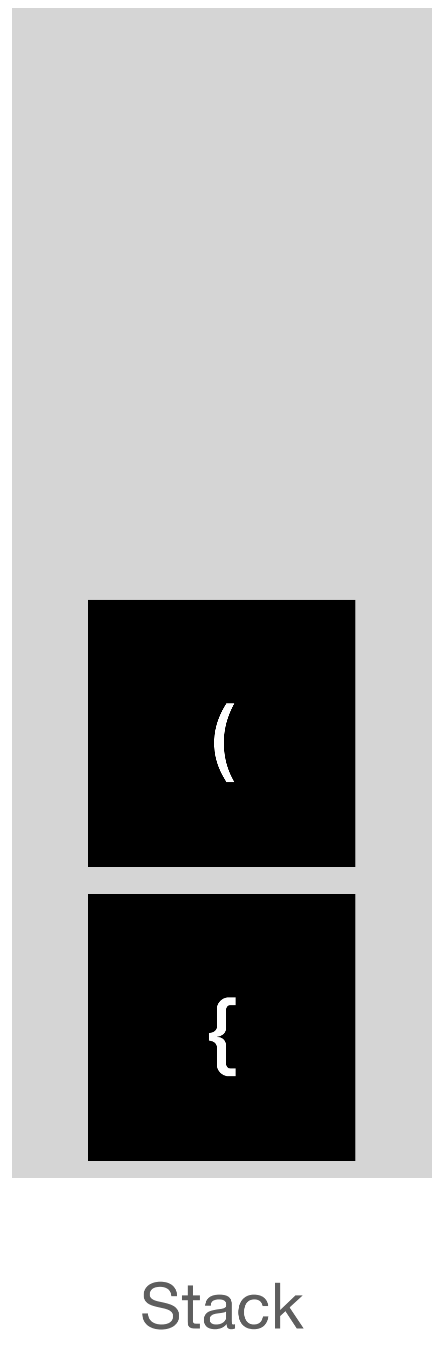
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};;}
                    ^
```

For each char in the string:

If it not a parenthesis, skip it…



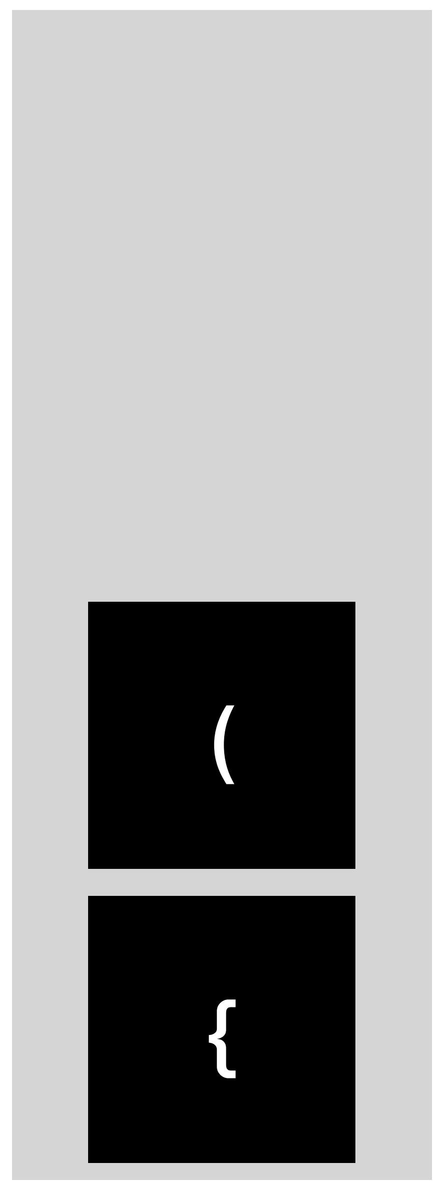
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
                    ^
```

For each char in the string:

If it not a parenthesis, skip it…



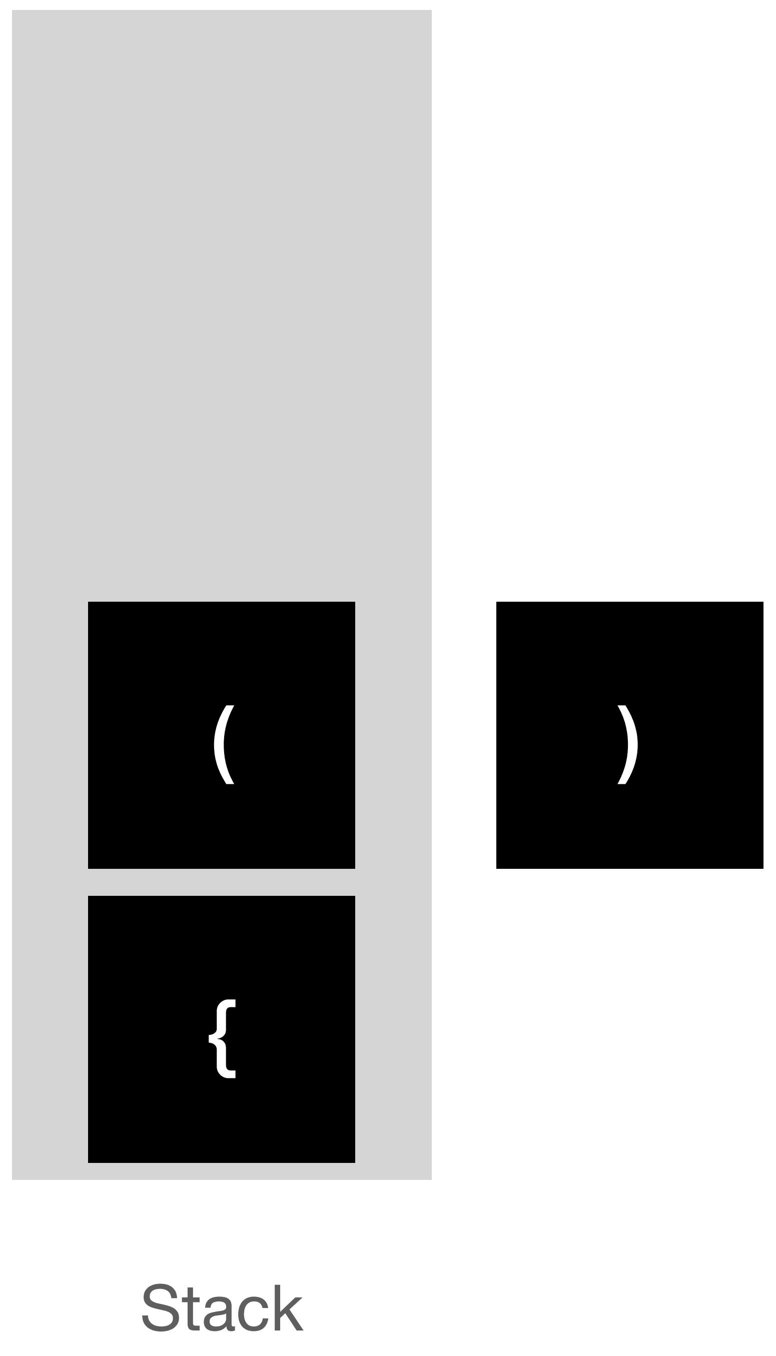
Stack

Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};;}
                       ^
```

```
For each char in the string:
    If it is a **right (closed) parenthesis**
        compare it with the top (and pop it)
            If they are not matched, return false
```



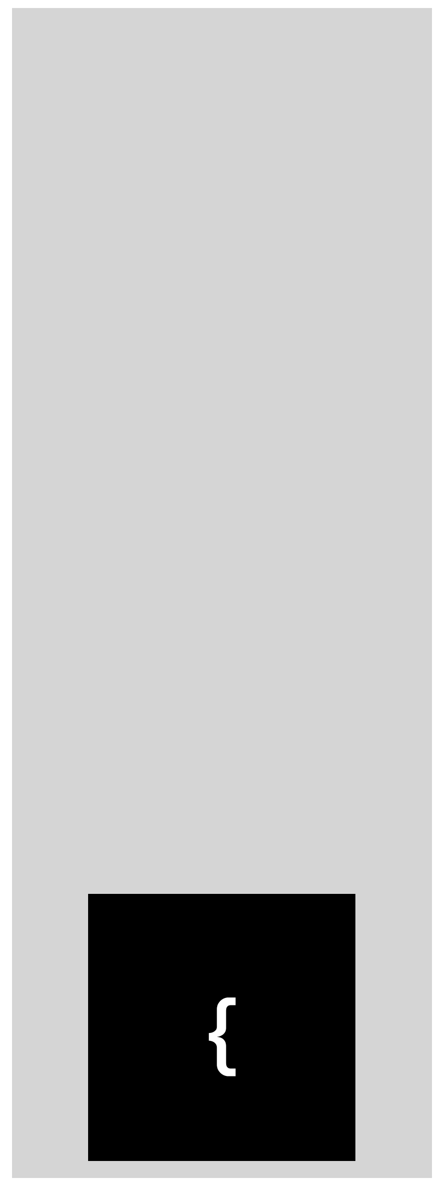
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
                         ^
```

For each char in the string:

If it not a parenthesis, skip it…



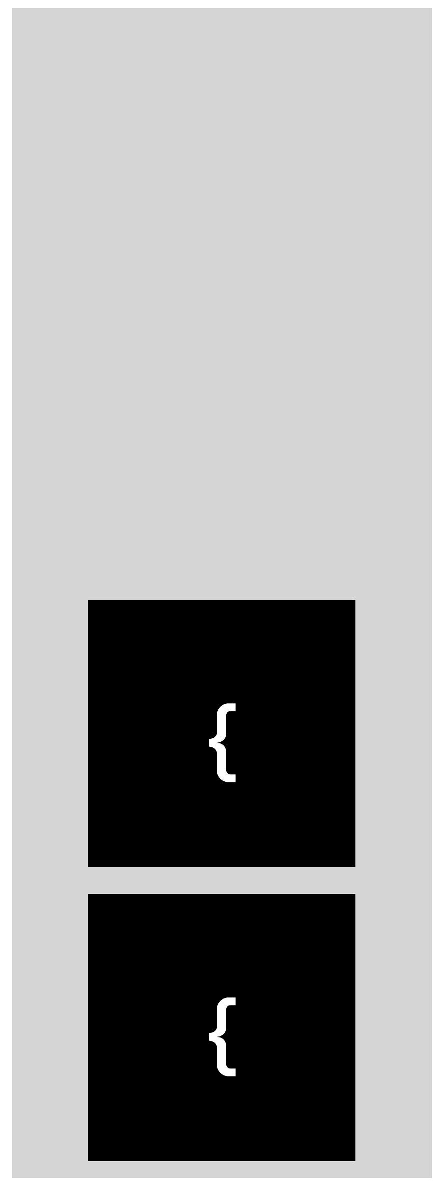
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
                          ^
```

For each char in the string:

If it not a parenthesis, skip it…



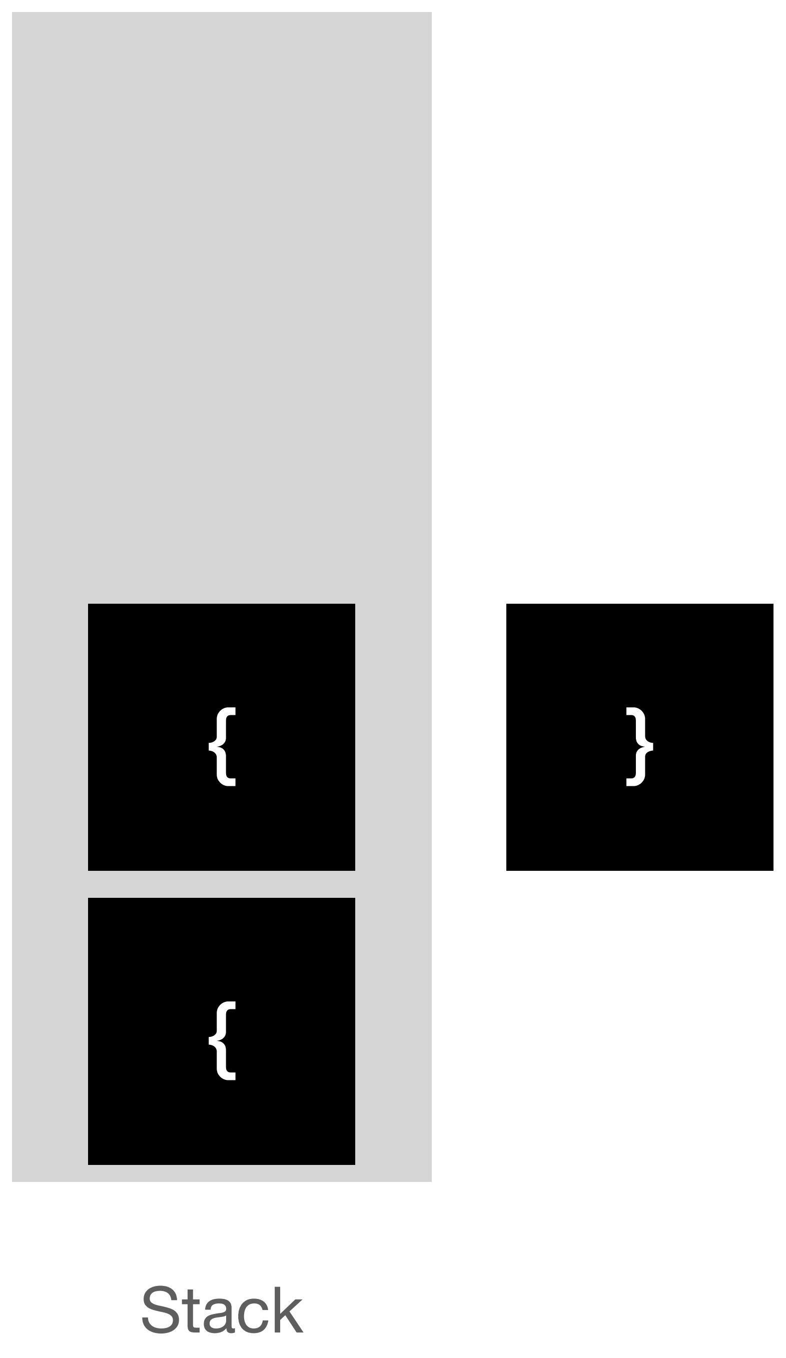
Stack

Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
                               ^
```

```
For each char in the string:
    If it is a right (closed) parenthesis
        compare it with the top (and pop it)
            If they are not matched, return false
```



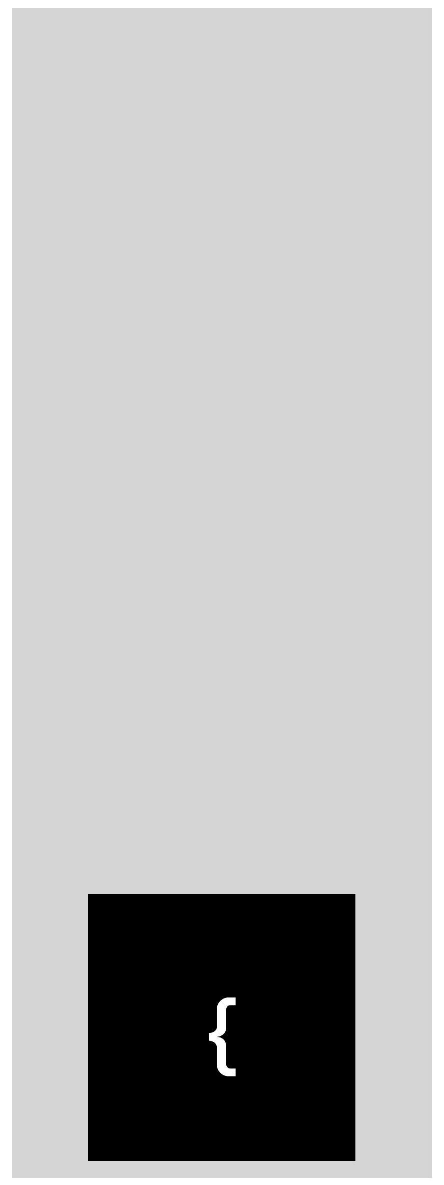
Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};}
                                 ^
```

For each char in the string:

If it not a parenthesis, skip it…



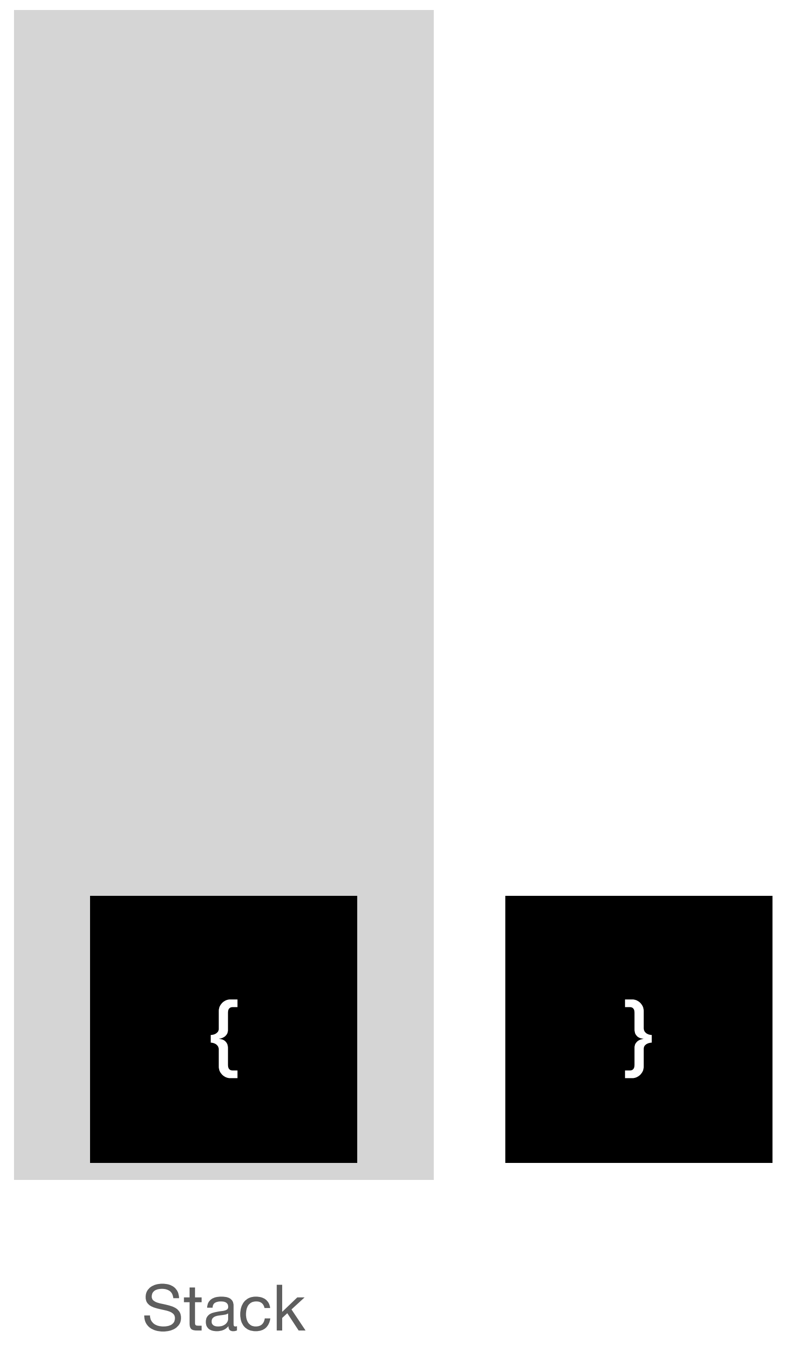
Stack

Balanced Parenthesis

Using stack!

```
void fun(){if (x[0] > 3) {y = 1};;}
                                  ^
```

```
For each char in the string:
    If it is a **right (closed) parenthesis**
        compare it with the top (and pop it)
            If they are not matched, return false
```

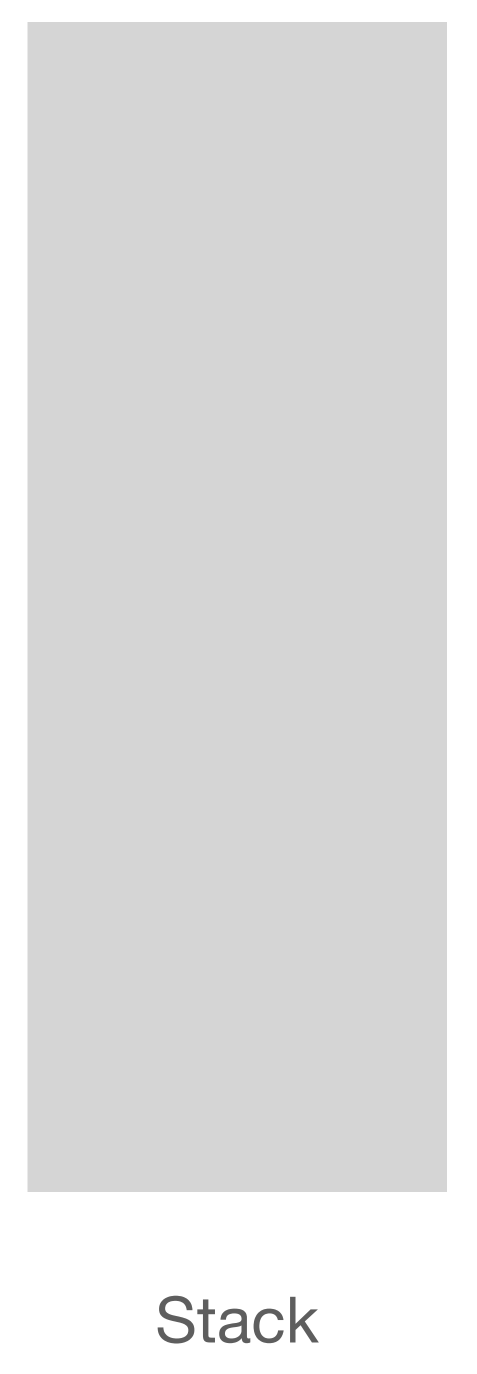


Balanced Parenthesis

Using stack!

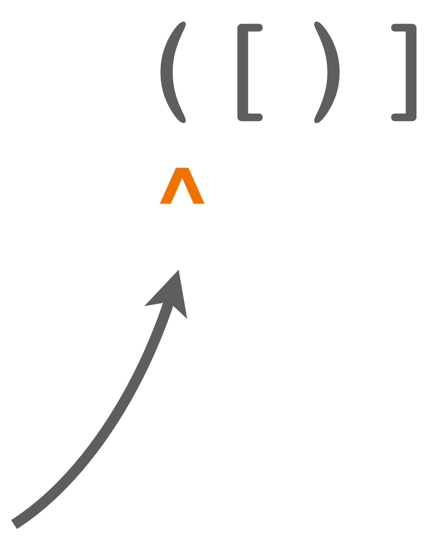
```
void fun(){if (x[0] > 3) {y = 1};;}
                                   ^
```

At the end, the stack should be empty



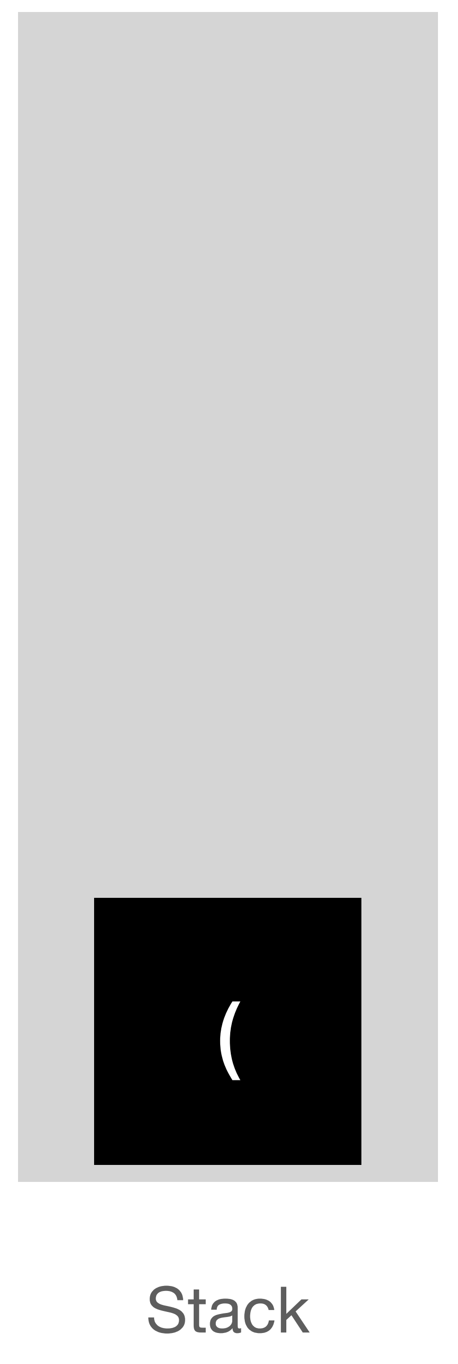
Balanced Parenthesis

Bad case



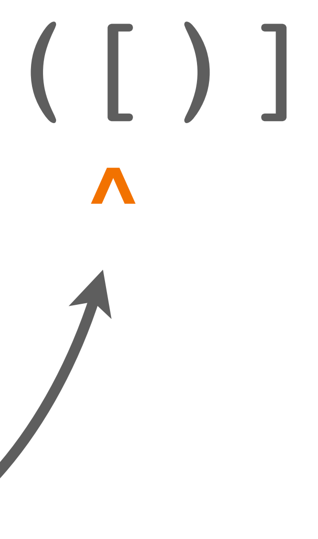
For each char in the string:

    If it is a **left (open) parenthesis,** push it on the stack



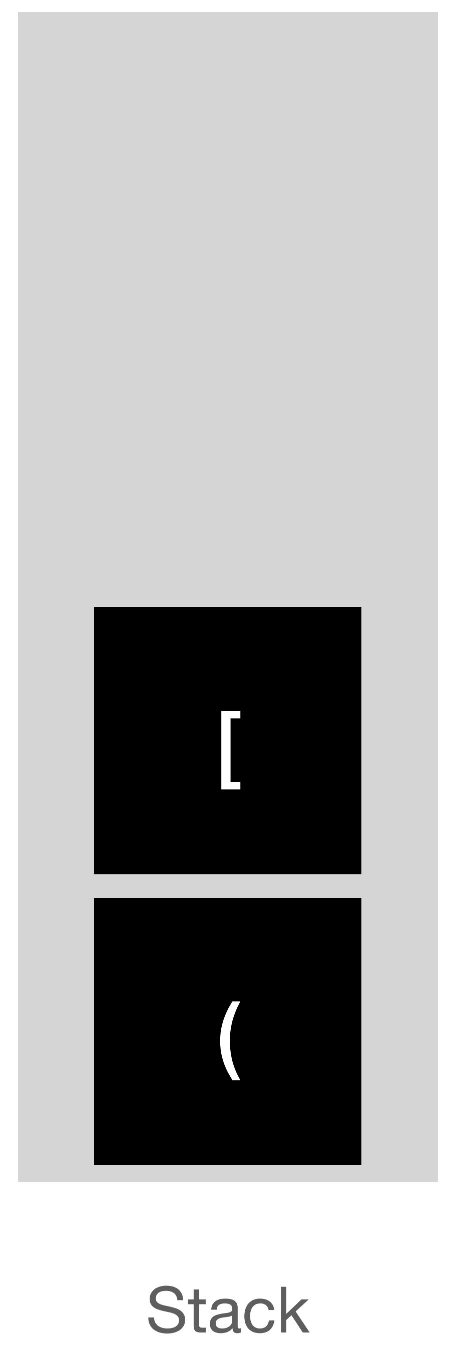
Balanced Parenthesis

Bad case



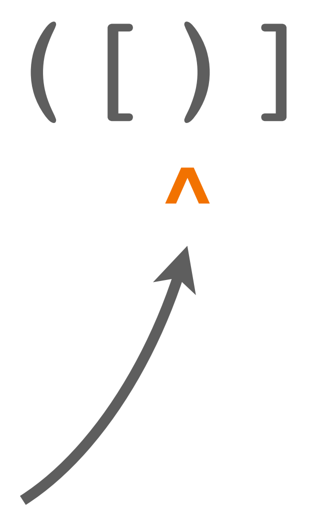
For each char in the string:

    If it is a **left (open) parenthesis,** push it on the stack

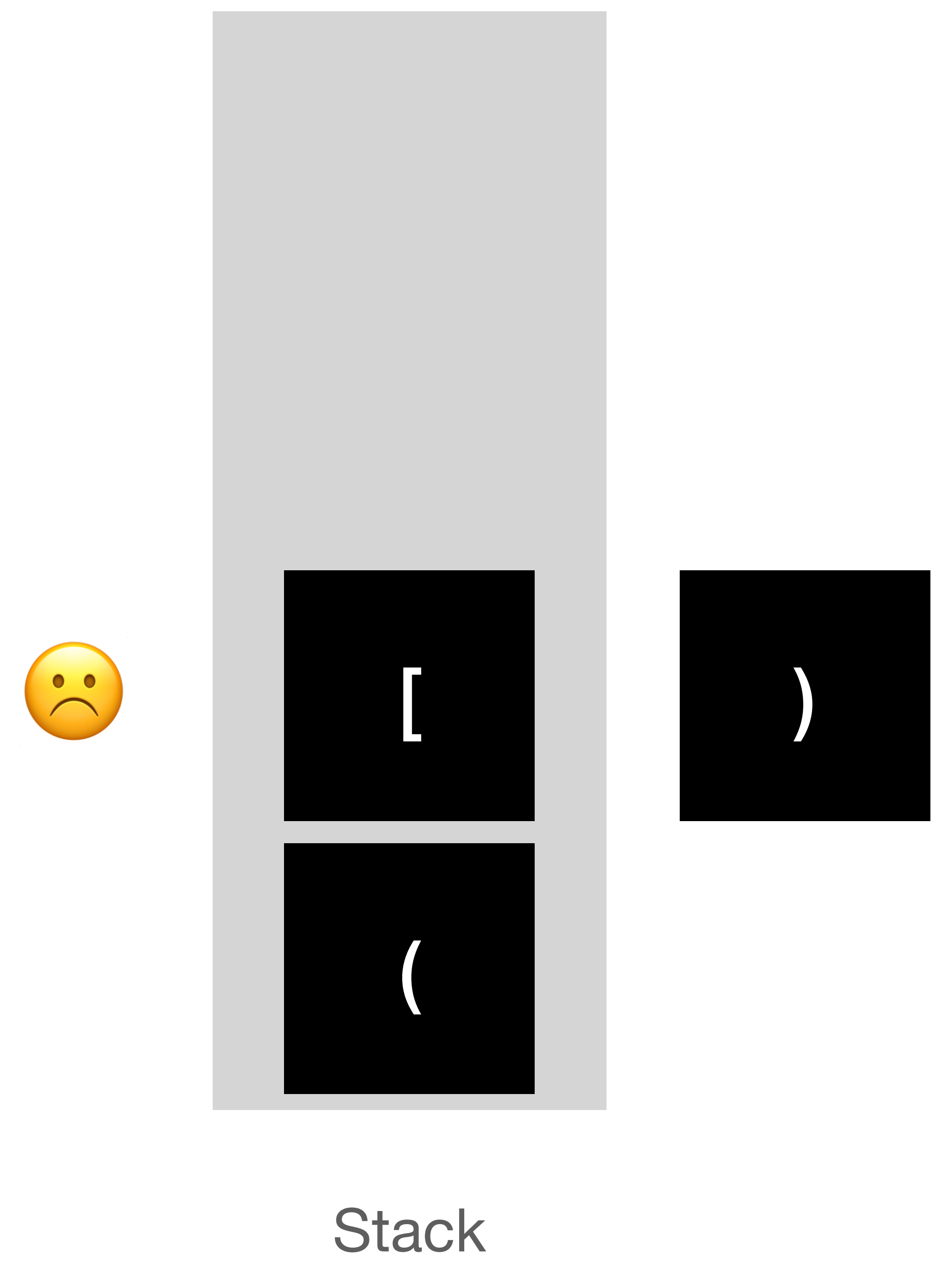


Balanced Parenthesis

Bad case

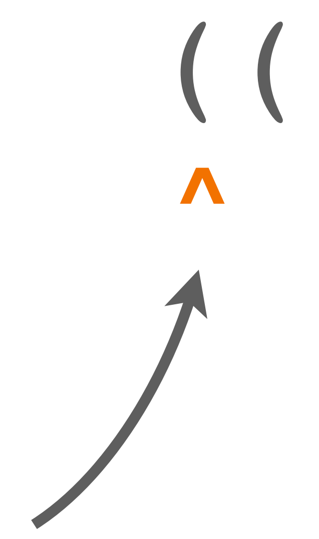


```
For each char in the string:
    If it is a right (closed) parenthesis
        compare it with the top (and pop it)
            If they are not matched, return false
```



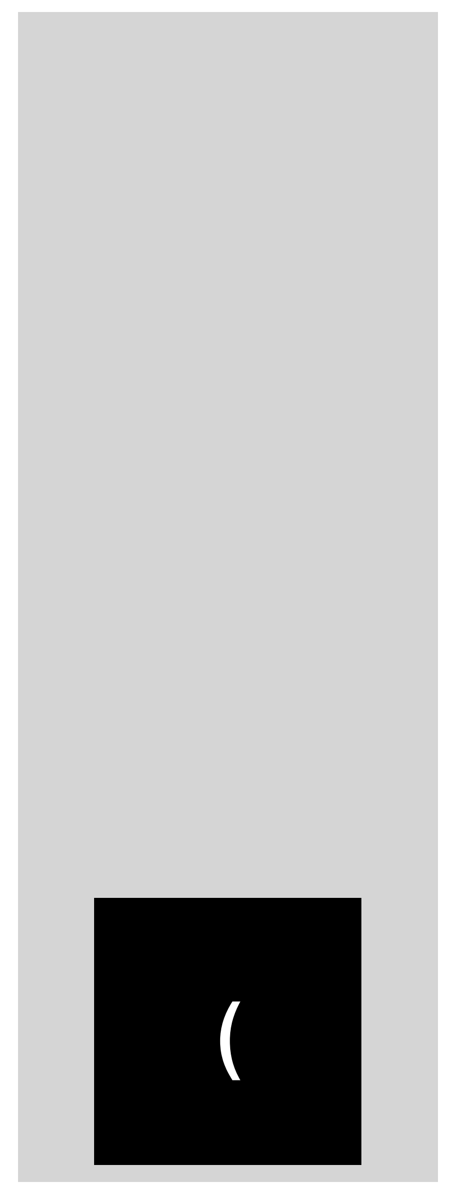
Balanced Parenthesis

Another bad case



For each char in the string:

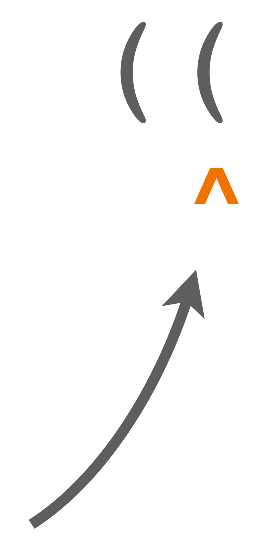
    If it is a **left (open) parenthesis,** push it on the stack



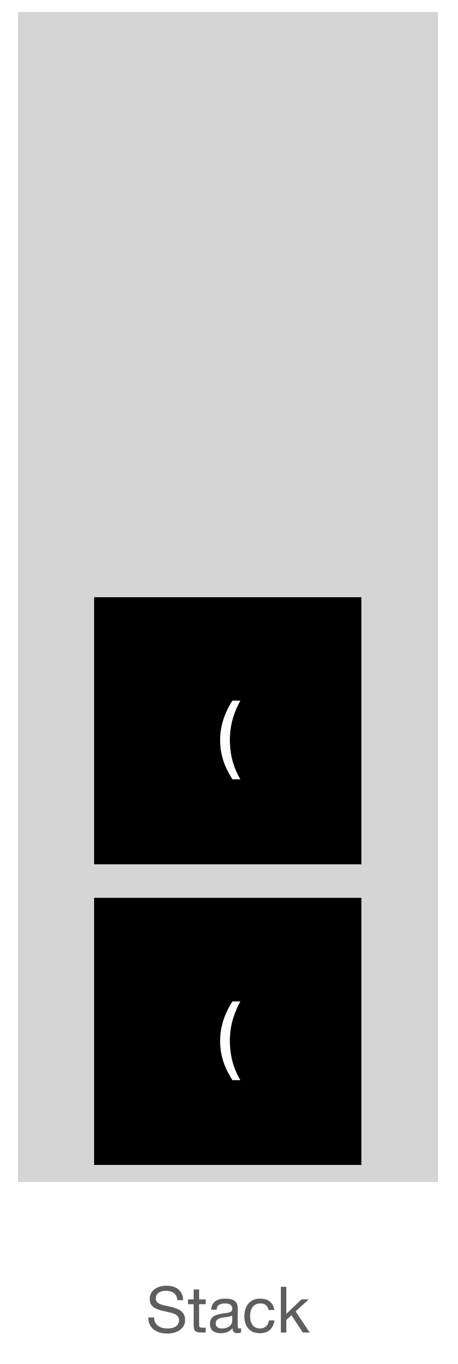
Stack

Balanced Parenthesis

Another bad case

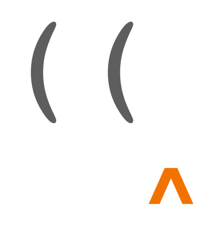


```
For each char in the string:
    If it is a **left (open) parenthesis,** push it on the stack
```

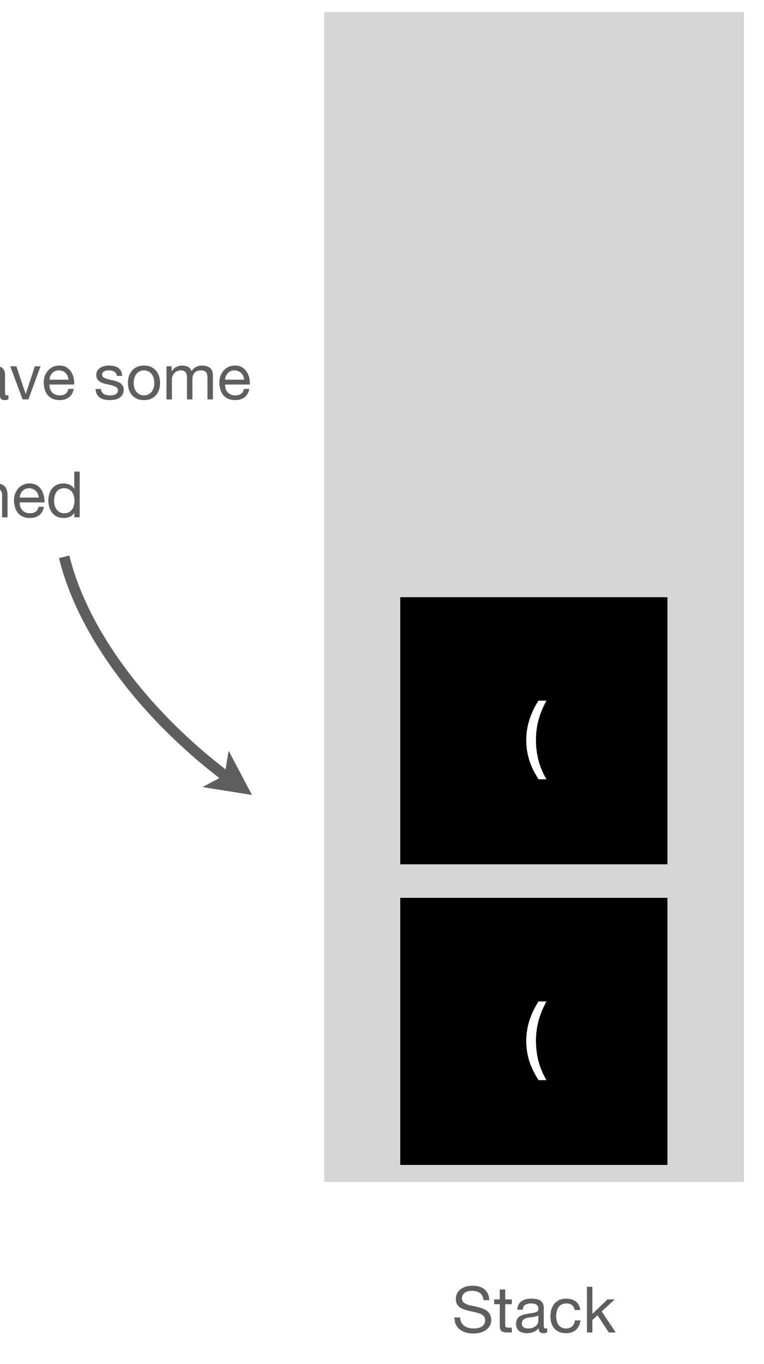


Balanced Parenthesis

Another bad case



☹️Oops! At the end, we still have some parenthesis that are not matched



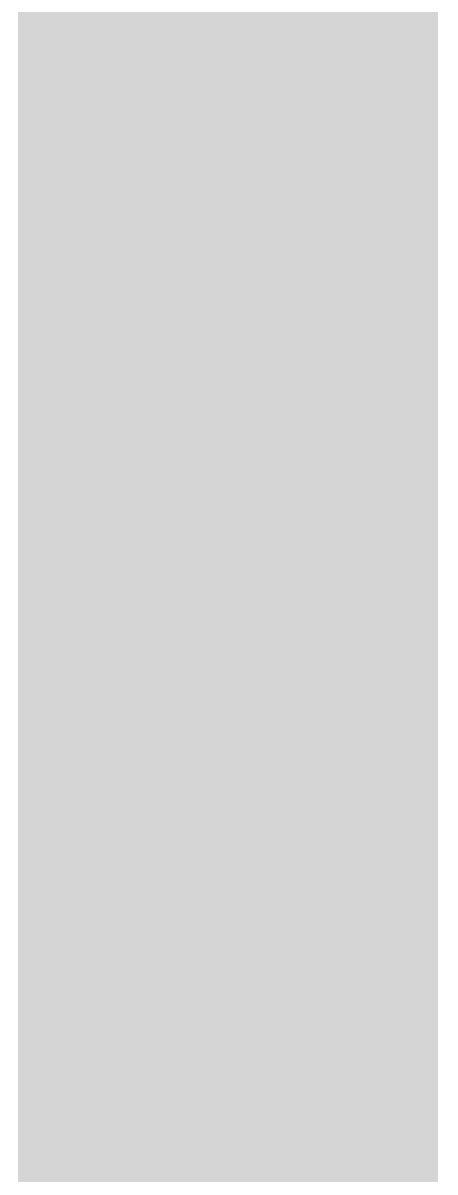
Balanced Parenthesis

Another bad case

```
23)
^
```

For each char in the string:

  If it not a parenthesis, skip it…



Balanced Parenthesis

Another bad case

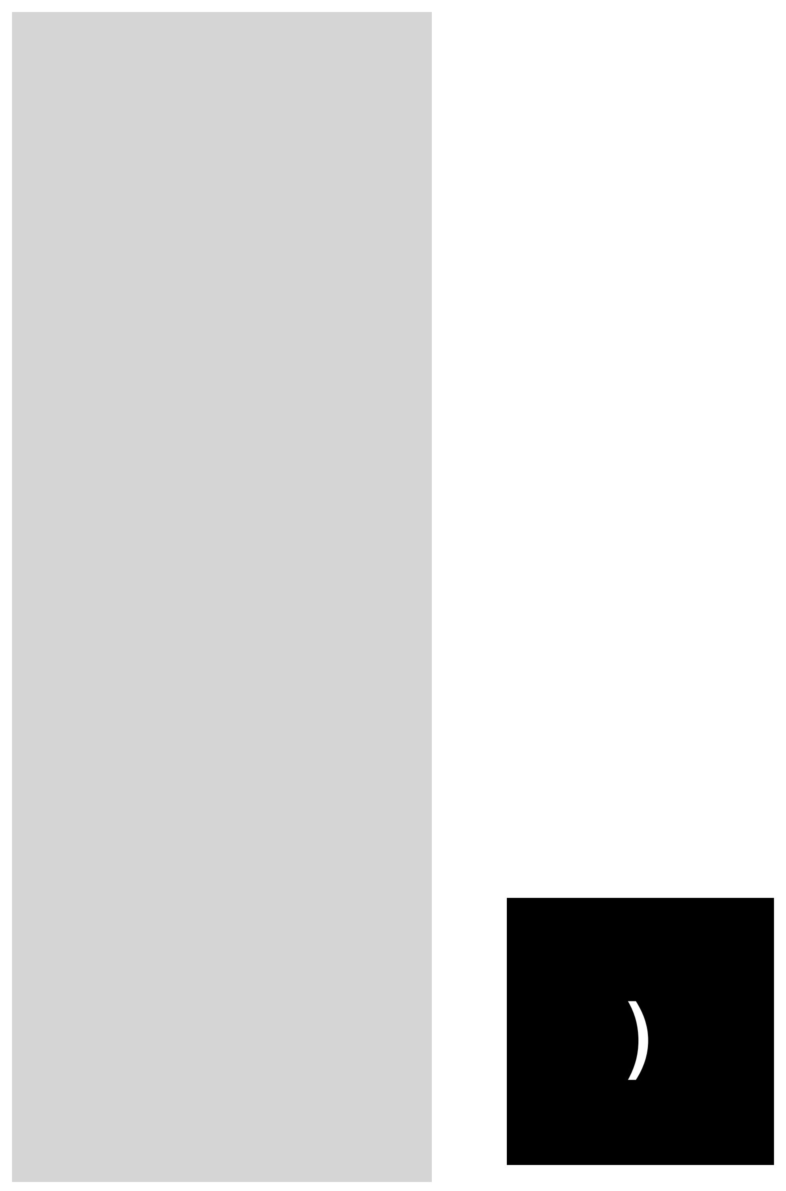
```
23)
  ^
```

```
For each char in the string:
    If it is a right (closed) parenthesis
        compare it with the top (and pop it)
            If they are not matched, return false
```

First, make sure it is not empty!

If empty, return false…

☹️ there is no matched left parenthesis in stack



Stack

Our algorithm

`isBalancedParenthesis()`

- For each char in the string:
  - If it is a left parenthesis, push it on top of the stack
  - If it is a right parenthesis:
    - If the stack is empty, return false
    - If it doesn’t pair the top value on the stack (remember to pop), return false
- At the end, if the stack is not empty, return false; otherwise, return true.

In-class problem

1. **Fork the following project**
2. **Get familiar with the code structures**
    - For simplicity, I merge all functions into a single file (not recommended, though)
3. **Finish the implementations of `isBalancedParenthesis()`**
    - You will use the three provided functions:
        - `isLeftParenthesis(char c): whether c is any left parenthesis, e.g.,{[(`
        - `isRightParenthesis(char c): whether c is any right parenthesis, e.g.,}])`
        - `isMatched(char l, char r): whether l and c are matched, e.g., { and }`
4. **Share the link of your finished project link to this Google sheet**