

# Collections, Part Two

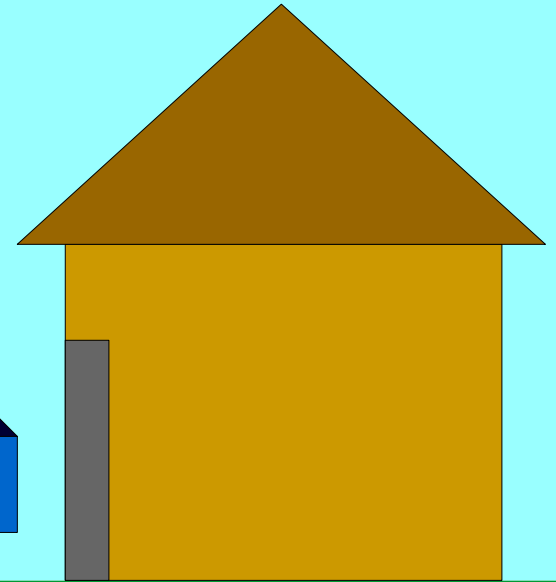
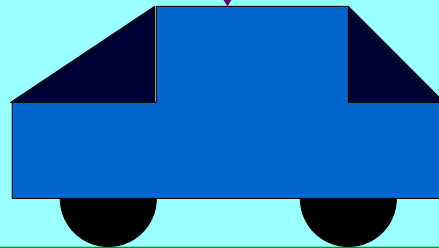
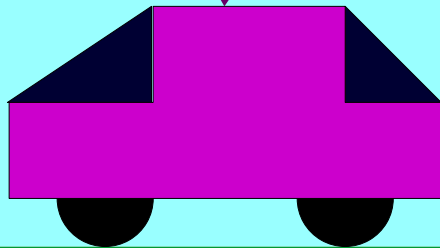
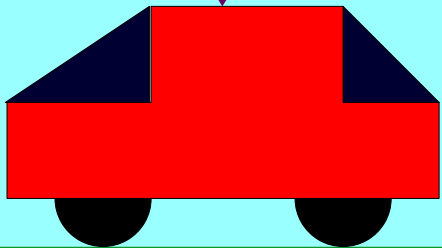
# Outline for Today

- ***Stacks***
  - Pancakes meets parsing!
- ***Queues***
  - Playing some music!

Stack

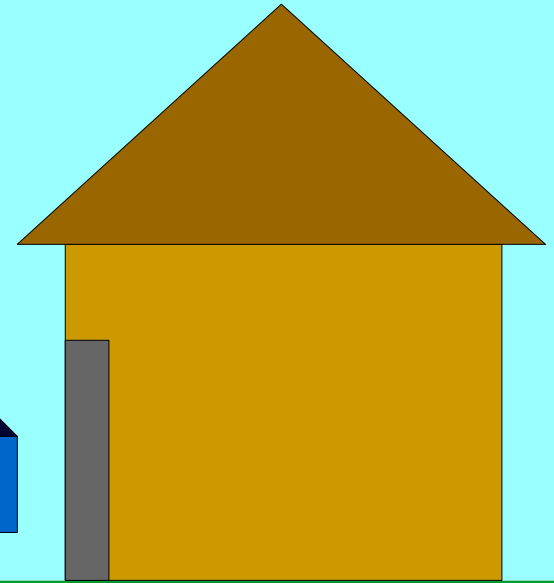
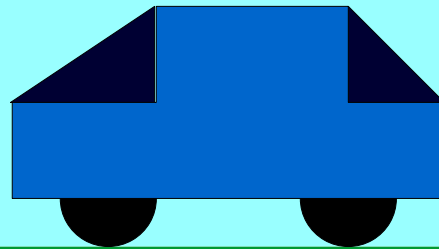
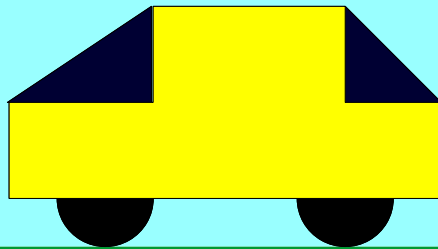
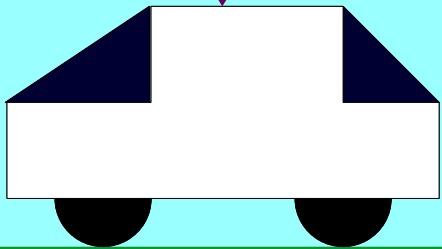
This car  
can't leave...

... until these  
two do.

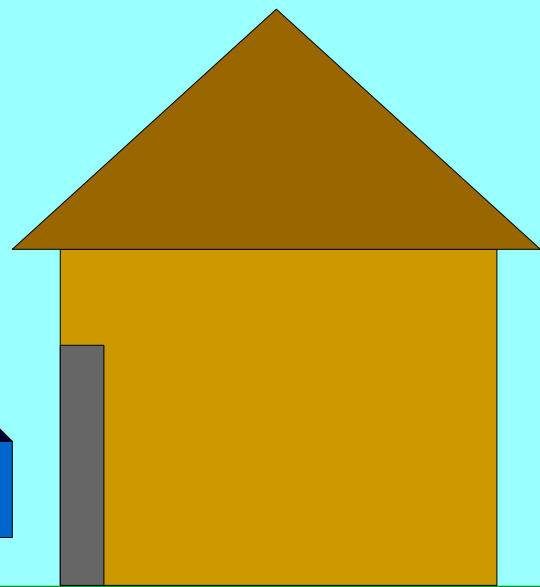
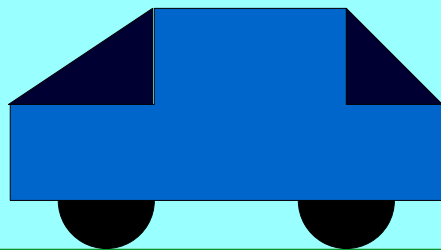
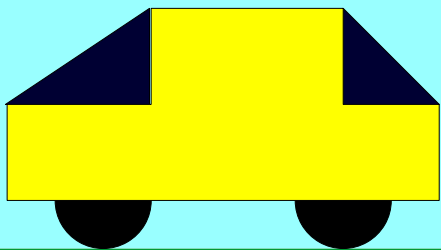
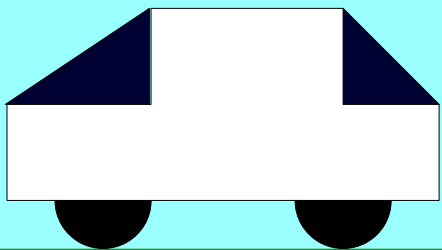


*Thanks to Nick Troccoli for this example!*

Any new car precedes all the old cars. Only this car can leave.



*Thanks to Nick Troccoli for this example!*



*Thanks to Nick Troccoli for this example!*

# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.





# Stack

137

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

42

137



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

271

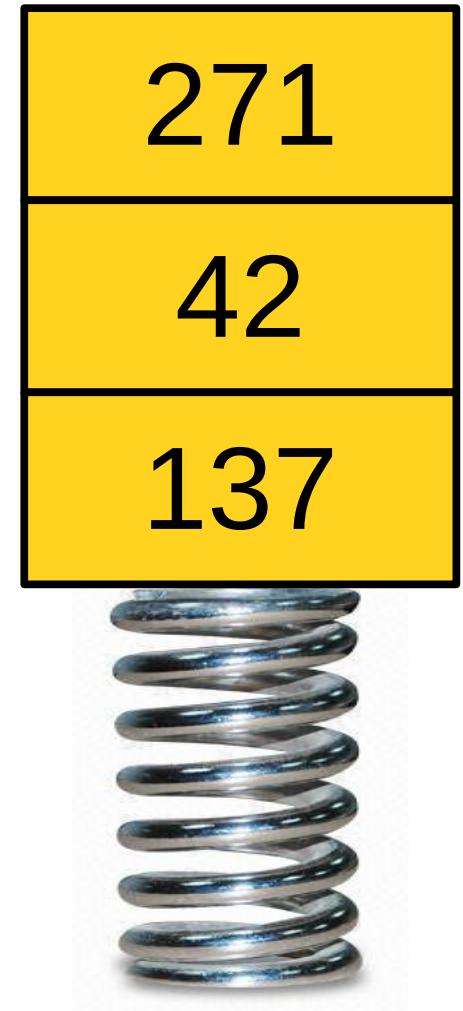
42

137



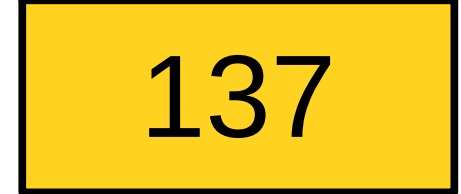
# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be *pushed* on top of the stack or *popped* from the top of the stack.





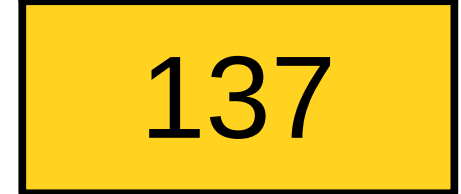
# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be *pushed* on top of the stack or *popped* from the top of the stack.



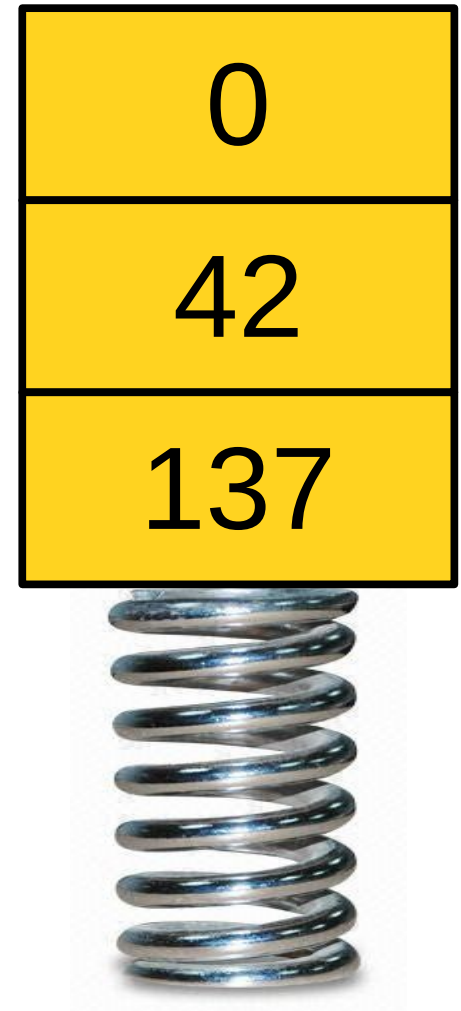
# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.



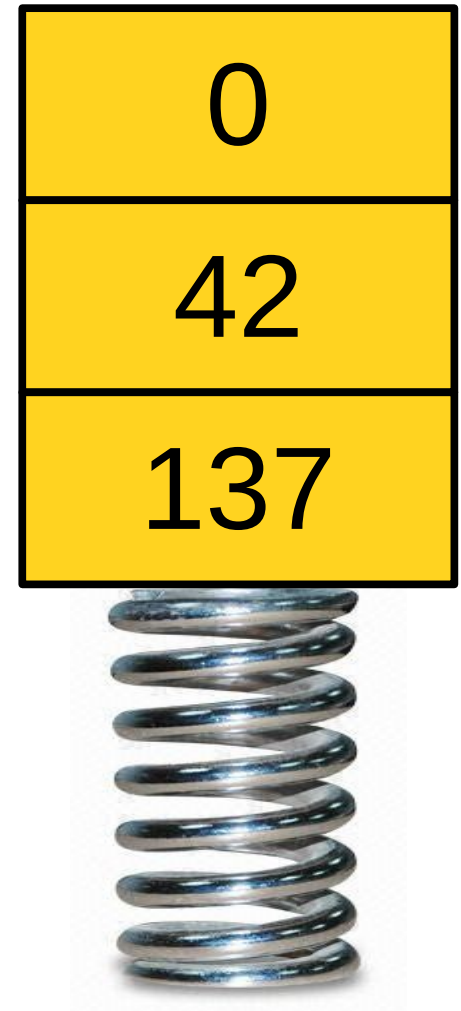
# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the topmost element of a Stack can be accessed.
- Do you see why we call it the *call stack* and talk about *stack frames*?



An Application: ***Balanced Parentheses***

# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }  
^
```

# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }  
^
```





# Balancing Parentheses

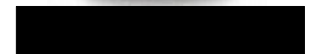
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

^



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



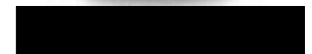
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



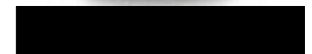
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo(^) { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo(^) { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo(^) { if (x * (y + z[1]) < 137) { x = 1; } }
```





# Balancing Parentheses

```
int foo(^) { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo(^) { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



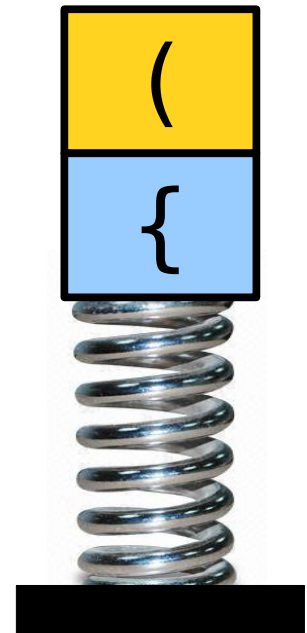
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



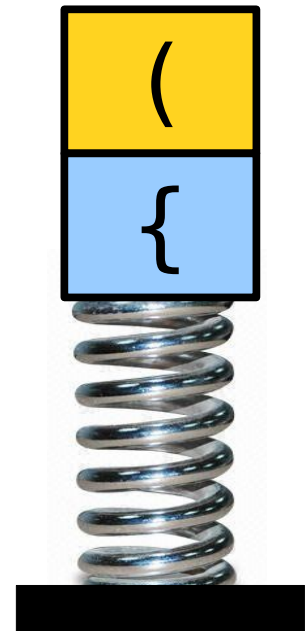
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



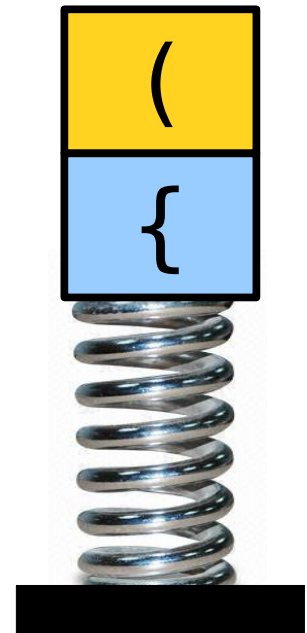
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



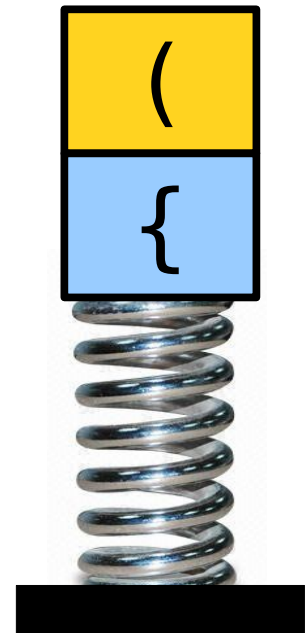
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



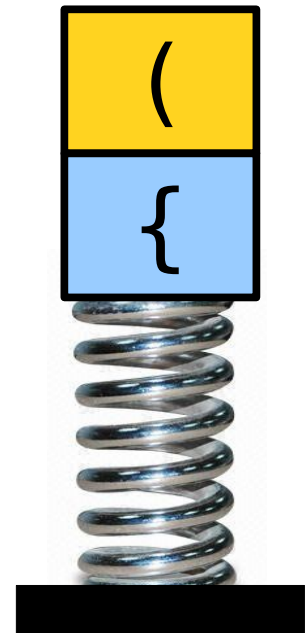
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



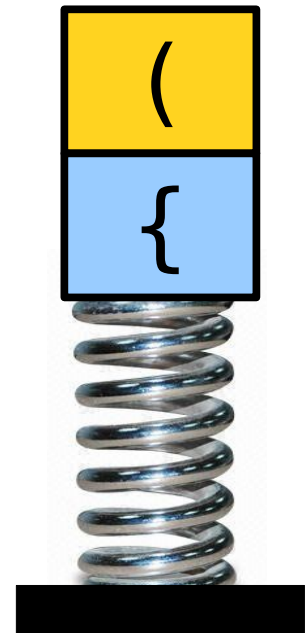
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

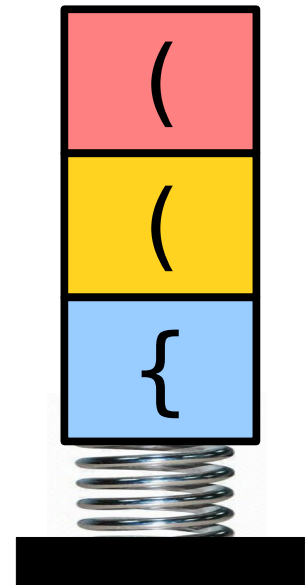
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





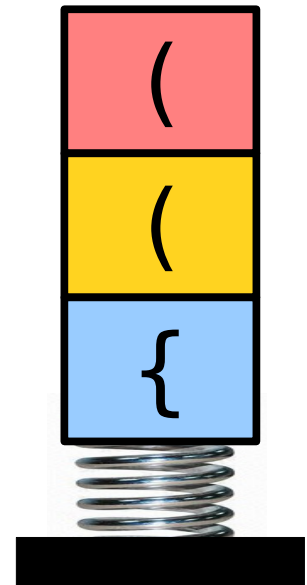
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



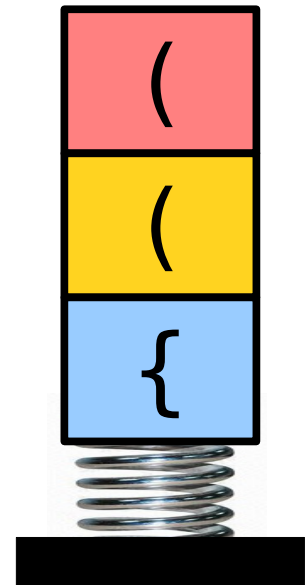
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



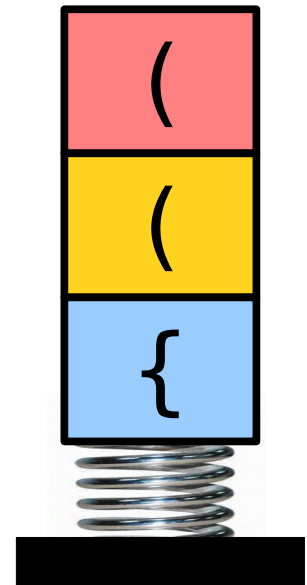
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



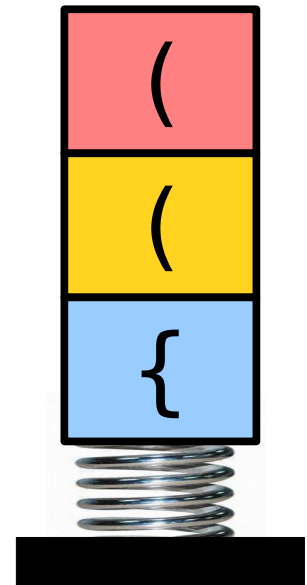
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



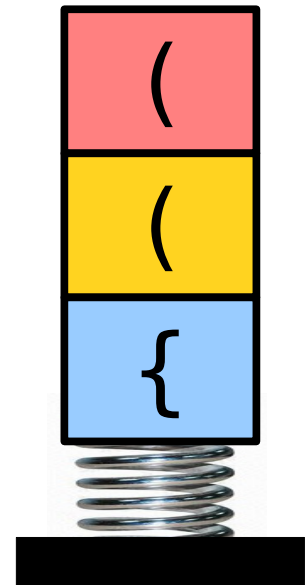
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



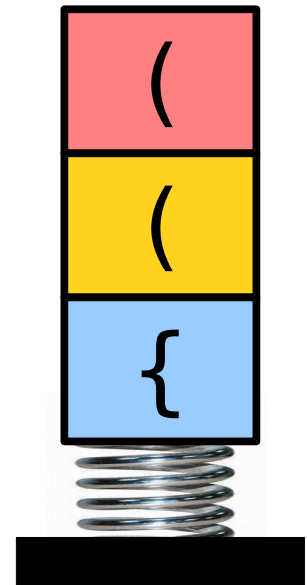
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



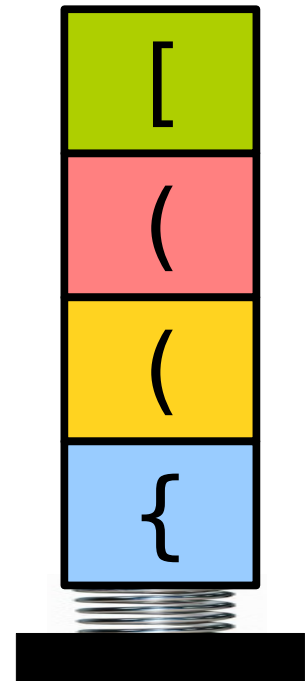
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

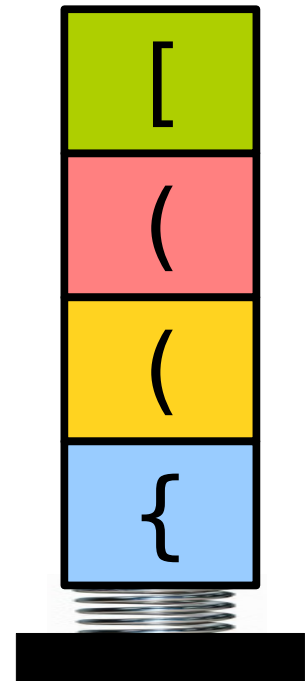
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





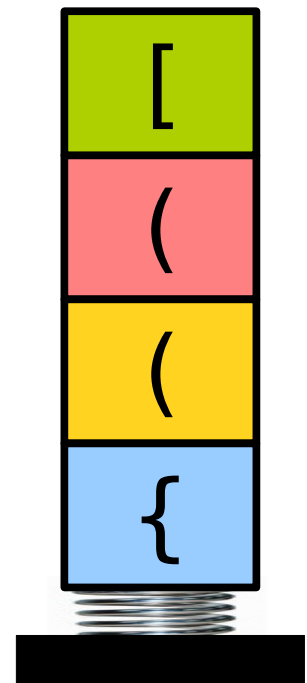
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



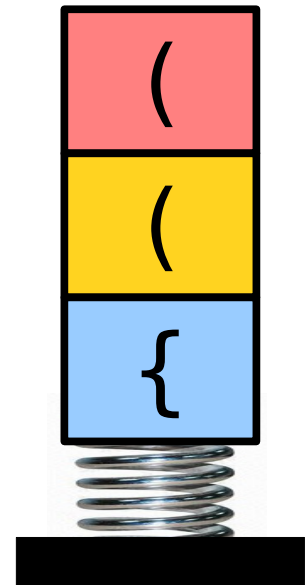
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



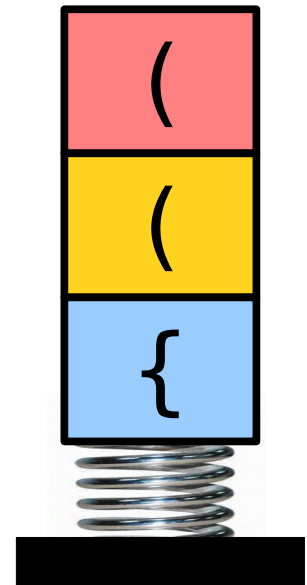
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



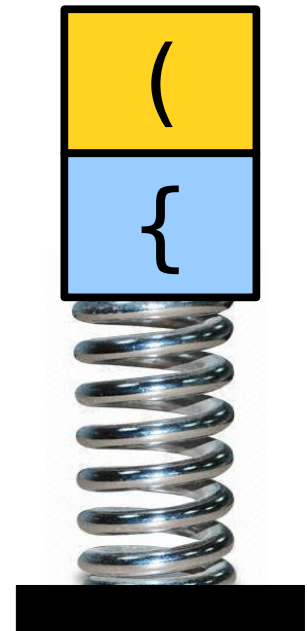
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



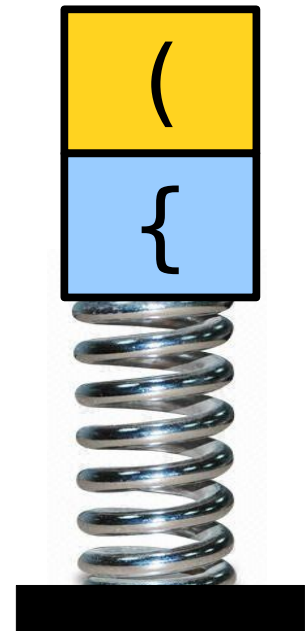
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



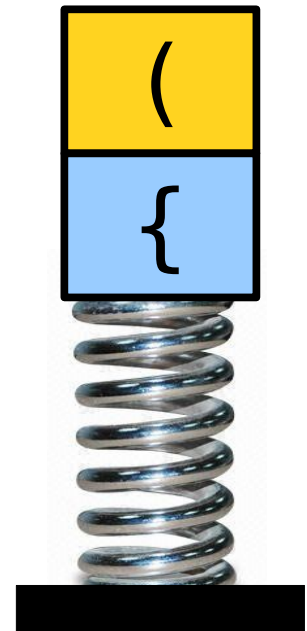
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



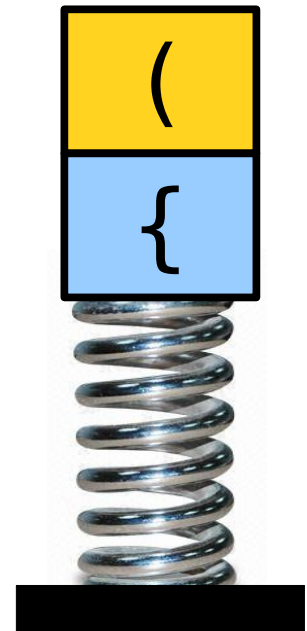
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

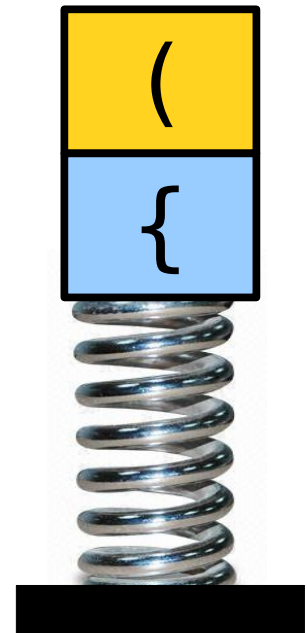
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





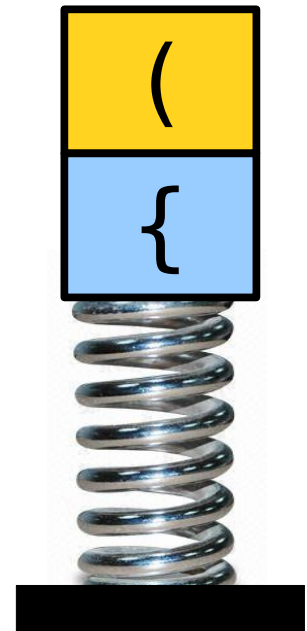
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



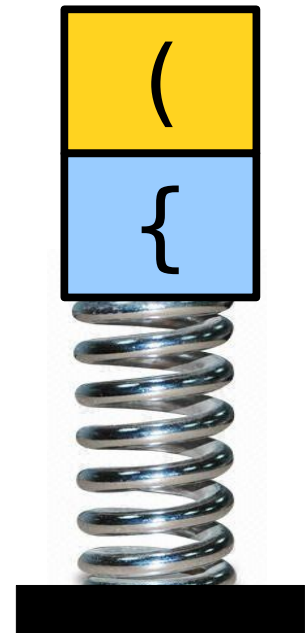
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



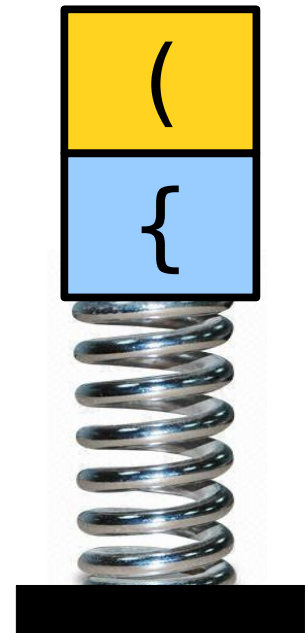
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



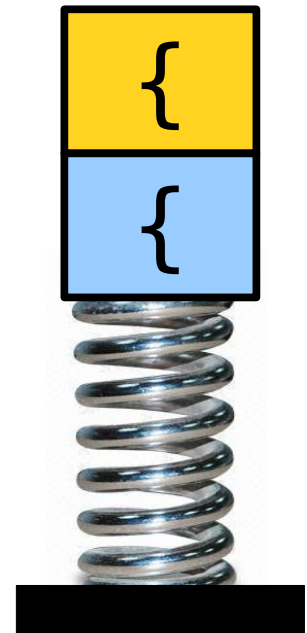
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

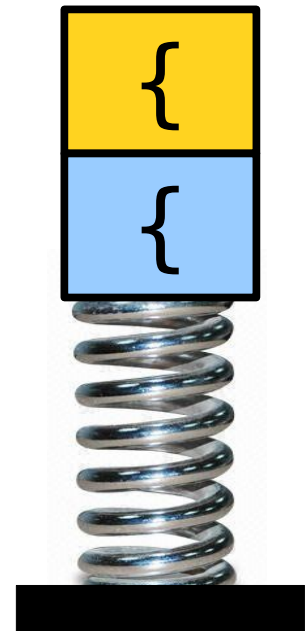
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





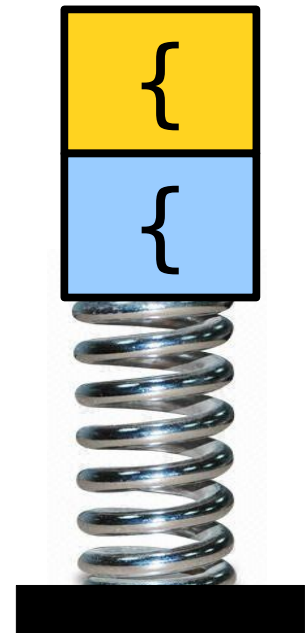
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



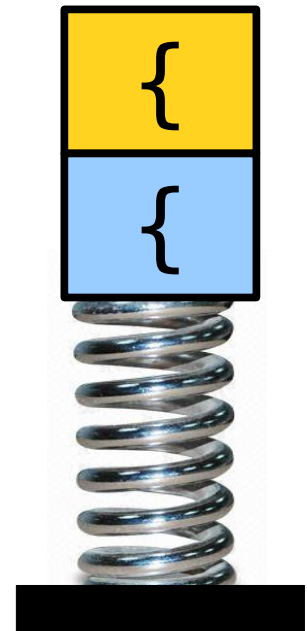
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



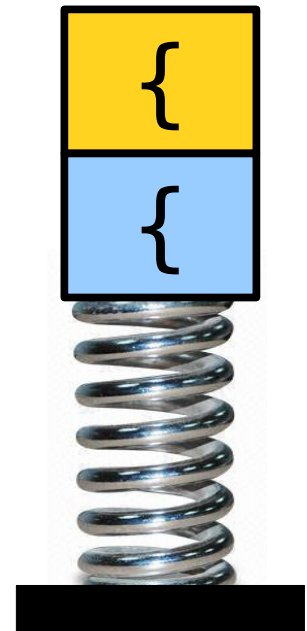
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



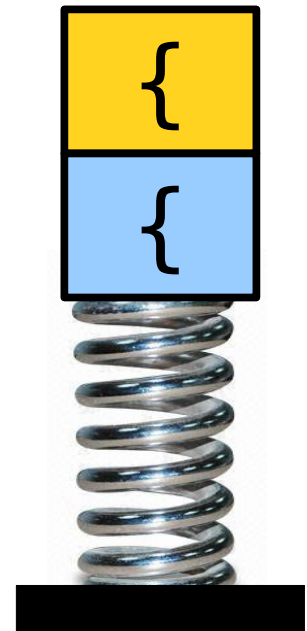
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



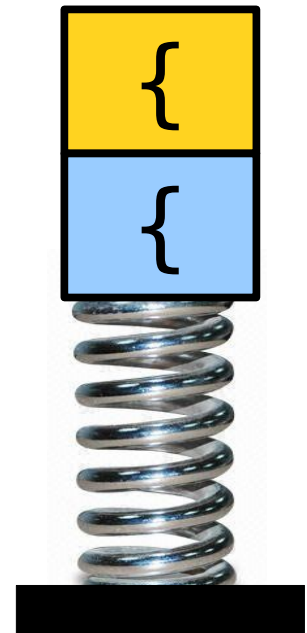
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



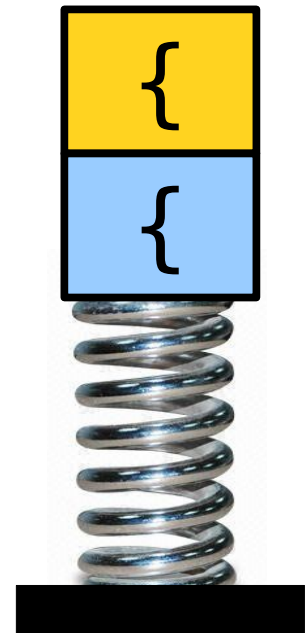
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



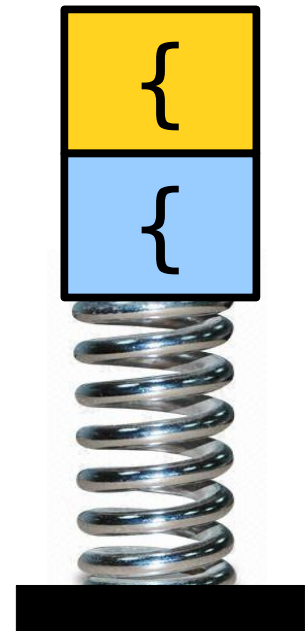
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

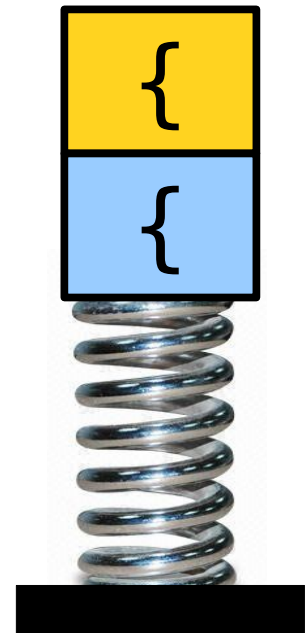
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }  
^
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }  
^
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } } ^
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

( [ ) ]





# Balancing Parentheses

( [ ) ]  
^



# Balancing Parentheses

( [ ) ]  
^



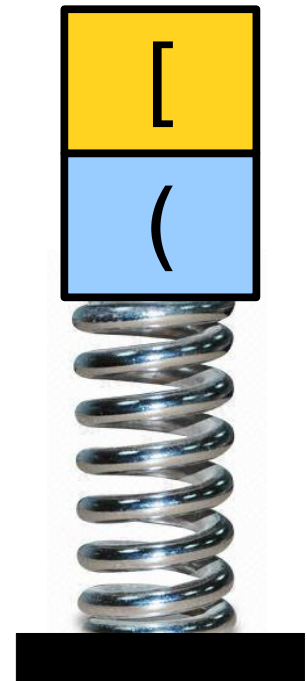
# Balancing Parentheses

( [ ) ]  
^



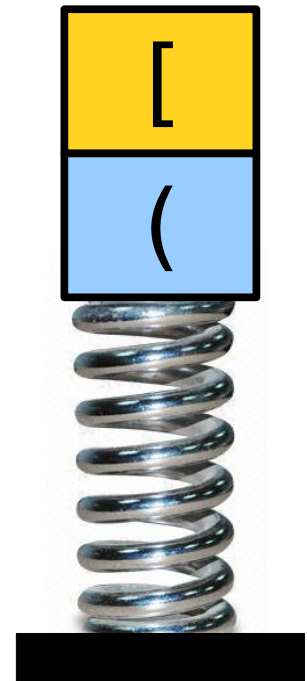
# Balancing Parentheses

( [ ) ]  
^



# Balancing Parentheses

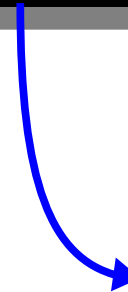
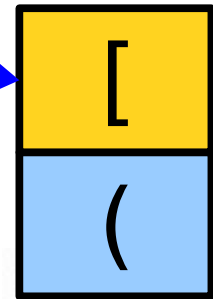
( [ ) ]  
          ^



# Balancing Parentheses

( [ ) ]  
          ^

Oops! Wrong type of  
parenthesis here.



# Balancing Parentheses

((



# Balancing Parentheses

( ( ^





# Balancing Parentheses

( (



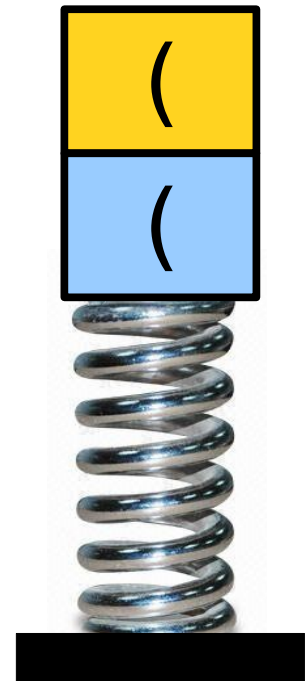
# Balancing Parentheses

( (



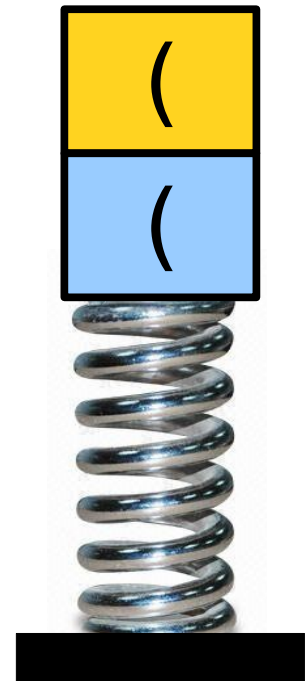
# Balancing Parentheses

( (



# Balancing Parentheses

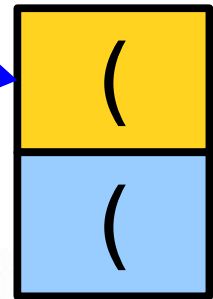
((



# Balancing Parentheses

( (

Oops! We never  
matched this.



# Balancing Parentheses

)



# Balancing Parentheses

)  
^



# Balancing Parentheses

Oops! There's  
nothing on the stack  
to match.

)  
^





# Our Algorithm

- For each character:
  - If it's an open parenthesis or brace, push it onto the stack.
  - If it's a close parenthesis or brace:
    - If the stack is empty, report an error.
    - If the character doesn't pair with the character on top of the stack, report an error.
- At the end, return whether the stack is empty (nothing was left unmatched).

# More Stack Applications

- Stacks show up all the time in *parsing*, recovering the structure in a piece of text.
  - Often used in natural language processing; take CS224N for details!
  - Used all the time in compilers – take CS143 for details!
  - There’s a deep theorem that says that many structures appearing in natural language are perfectly modeled by operations on stacks; come talk to me after class if you’re curious!
- They’re also used as building blocks in larger algorithms for doing things like
  - making sure a city’s road networks are navigable (finding *strongly connected components*; take CS161 for details!) and
  - searching for the best solution to a problem – stay tuned!

**Time-Out for Announcements!**

# Assignment 1

- Assignment 1 is due this Friday at the start of class.
- Have questions?
  - Call into the LaIR!
  - Ask on EdStem!
  - Email your section leader!

# Discussion Sections

- Discussion sections have started! You should have received an email with your section time and section leader's name.
- Don't have a section? You can sign up for any open section by visiting

[cs198.stanford.edu](http://cs198.stanford.edu)

logging in via “CS106 Sections Login,” and picking a section of your choice.

# Get Involved in Research!

Applications for Summer Computer Science Research Internships (CURIS) at Stanford open on Sunday, January 24th. CURIS is the Computer Science department's undergraduate research program: Stanford students work with Ph.D. students and a faculty member for the Summer, with the goal of learning about and contributing to research. Many internships produce an identifiable and publishable research result, with students going on to top Ph.D. programs. Students who participate in CURIS open up opportunities in top-tier jobs, graduate schools, and often see their work shared and adopted worldwide.

CURIS stipends are \$5,000-\$9,000 for 10 weeks. The stipend level is set by the University following the guidelines on this page (go down to Dynamic Stipend Overview):

<https://undergrad.stanford.edu/opportunities/research/information-faculty-staff/faculty-department-grant-administration>

Apply at <http://curis.stanford.edu>. Applications are due by Sunday, February 7. You rank the projects you've applied to, research groups rank the students who applied to their projects, and CURIS matches students to projects using the Gale-Shapely bipartite matching algorithm.

Projects hold office hours (over Zoom) during the application period so you can talk with the faculty and students on the project to introduce yourself and learn more. Project listings should specify their office hours; if they aren't listed, email the contact person.

To help with your application, the CURIS mentors are holding office hours.

Tuesdays: 1-2pm - Book a slot at <https://calendly.com/aabuhash/curisoh>

Tuesdays: 4-5pm (drop-in) - Zoom Link

There will also be an information session with Ph.D. students and prior CURIS participants, which we will announce later this week.

For any additional question, please email [curis-mentors@cs.stanford.edu](mailto:curis-mentors@cs.stanford.edu)

# Apply for Internships!

The Computer Forum will be holding their Winter Career Fair virtually on [Career Fair Plus](#) on January 27, from 11:00am - 4:00pm. This fair is only for Stanford students.

Computer Forum Career Fair:

Date: Wednesday, January 27

Time: 11:00am - 4:00pm PT

Student registration is now open and you can book your meetings with companies [HERE](#).

[Here is a helpful article on how to prepare for the career fair](#)

In addition, you can [read this article](#) on how to book meeting on the web portal.

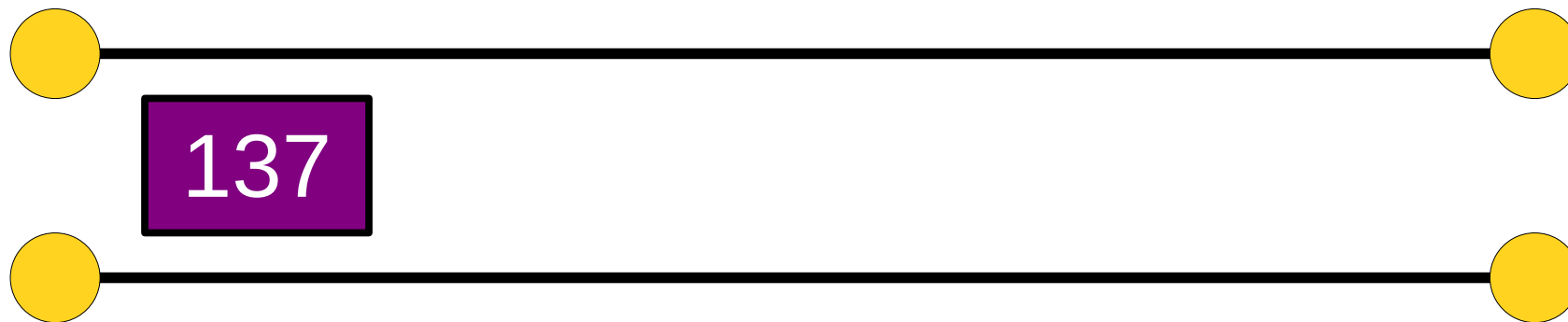
```
lecture.pop();
```



Queue

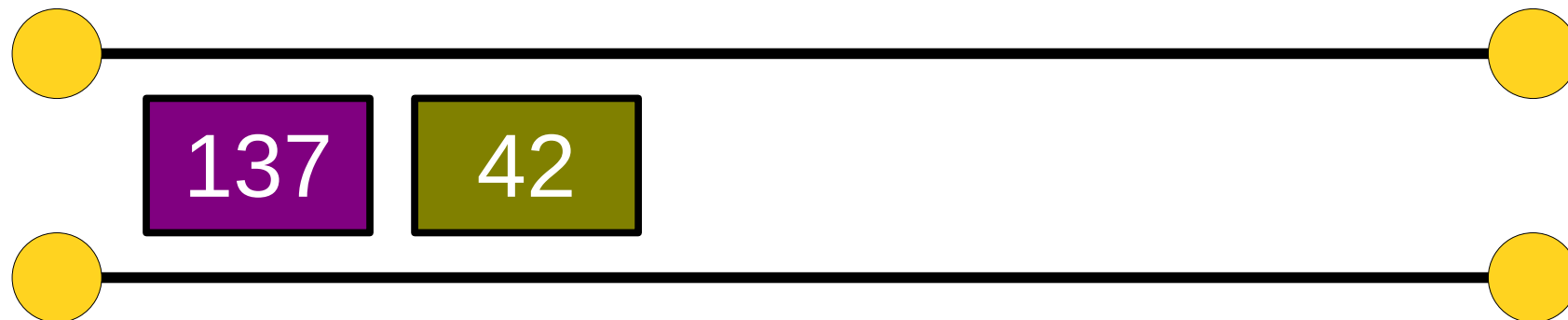
# Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



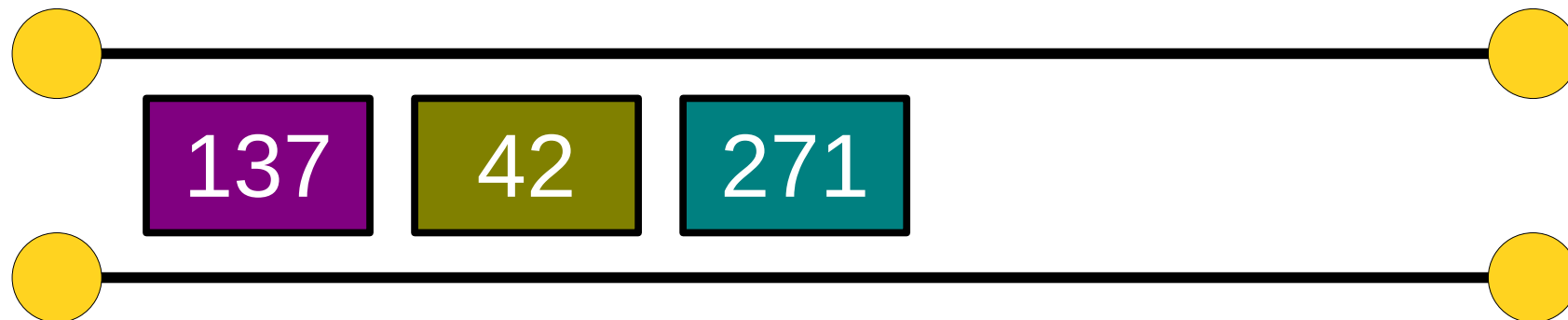
# Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



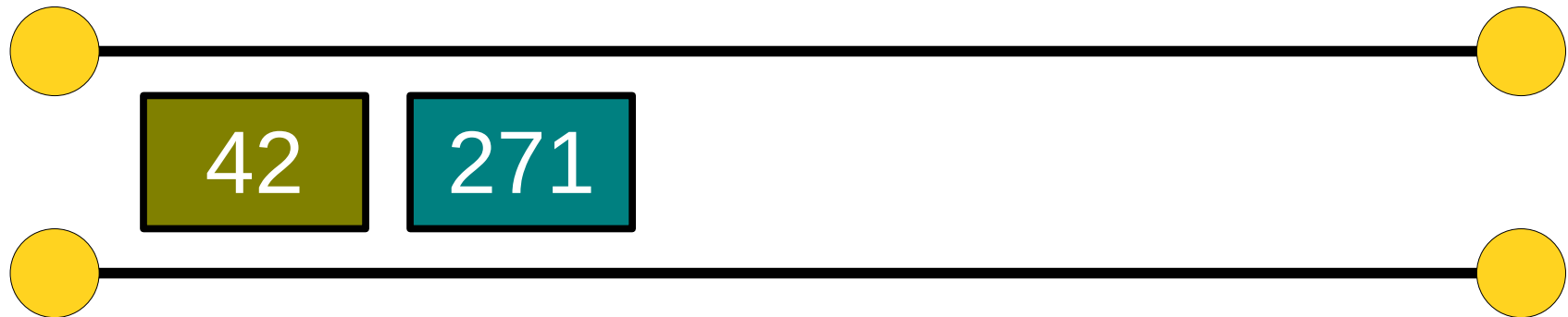
# Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



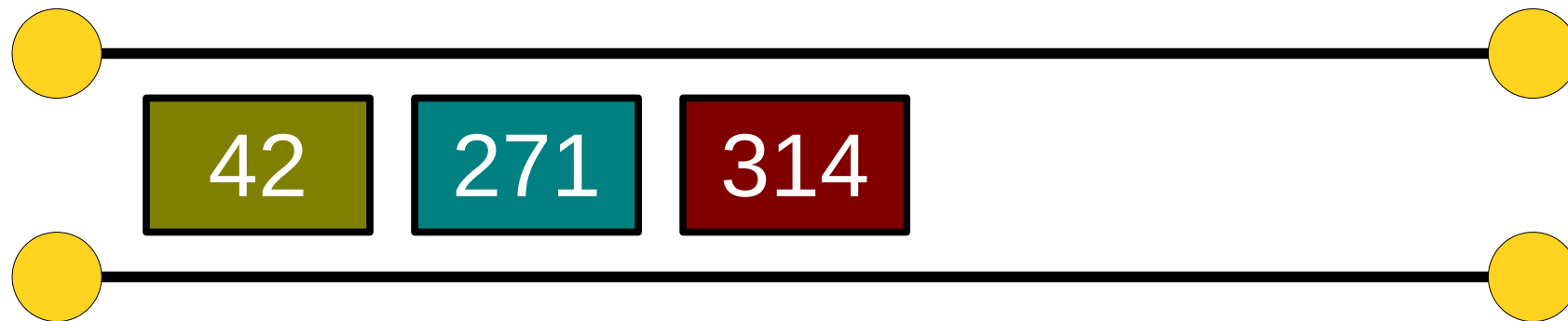
# Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



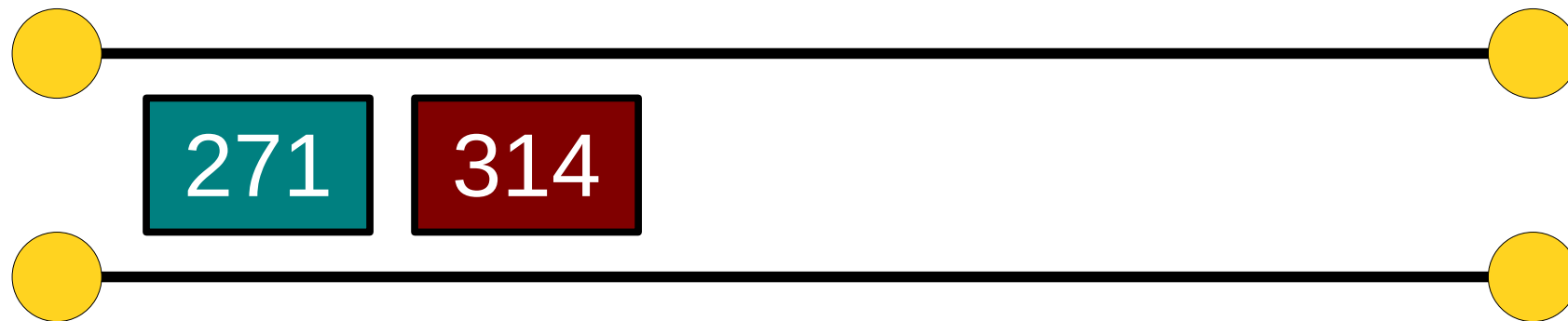
# Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



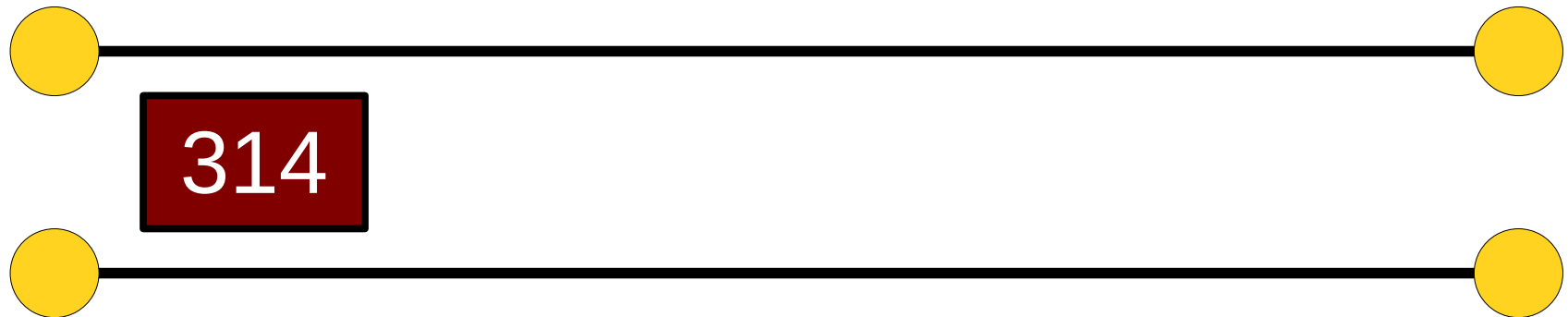
# Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



# Queue

- A **Queue** is a data structure representing a waiting line.
- Objects can be **enqueued** to the back of the line or **dequeued** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.





An Application: *Looper*

# Loopers

- A *looper* is a device that records sound or music, then plays it back over and over again (in a loop).
- These things are way too much fun, *especially* if you're not a very good musician.
- Let's make a simple looper using a Queue.

# Building our Looper

- Our looper will read data files like the one shown to the left.
- Each line consists of the name of a sound file to play, along with how many milliseconds to play that sound for.
- We'll store each line using the `SoundClip` type, which is defined in our C++ file.

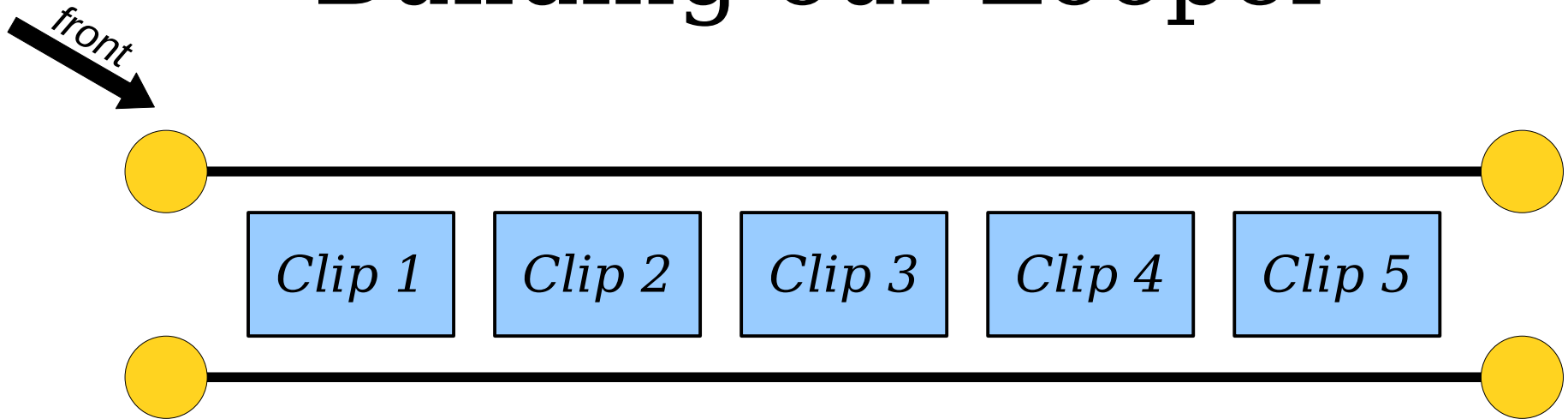
```
G2.wav 690
G2.wav 230
Bb2.wav 230
G2.wav 460
G2.wav 460
G2.wav 460
G2.wav 230
Bb2.wav 230
G2.wav 230
F2.wav 460
```

# Building our Looper

# Building our Looper

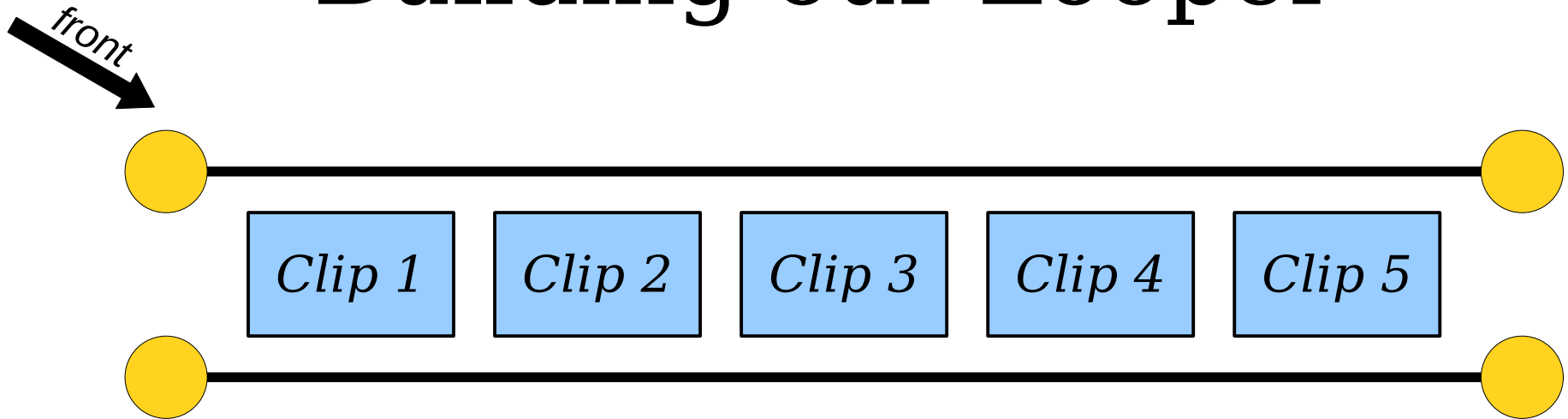
```
Queue<SoundClip> loop = loadLoop(/* ... */);
```

# Building our Looper



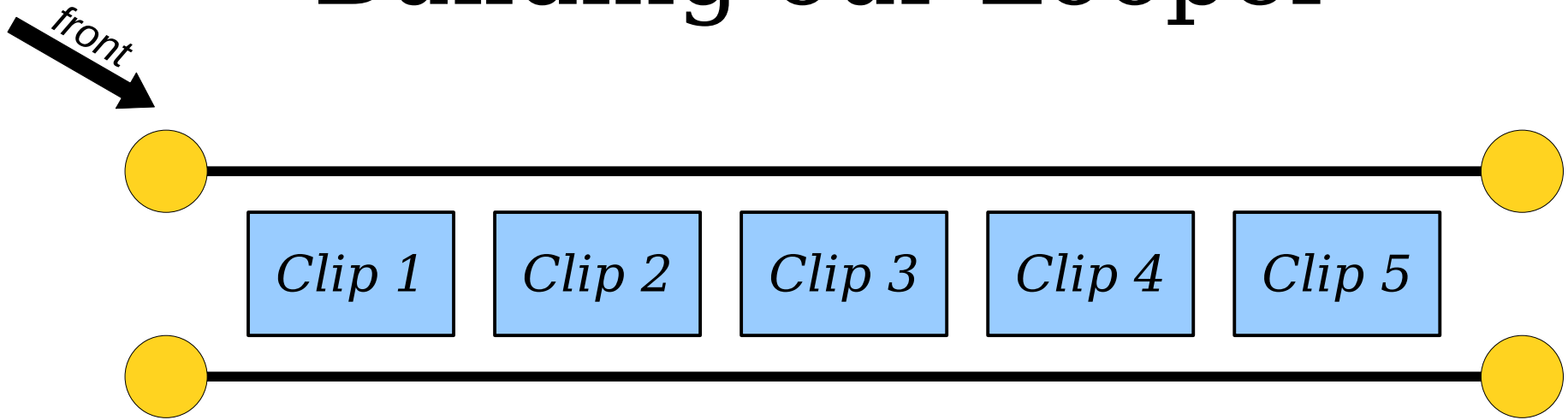
```
Queue<SoundClip> loop = loadLoop(/* ... */);
```

# Building our Looper



```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
  
  
  
  
}
```

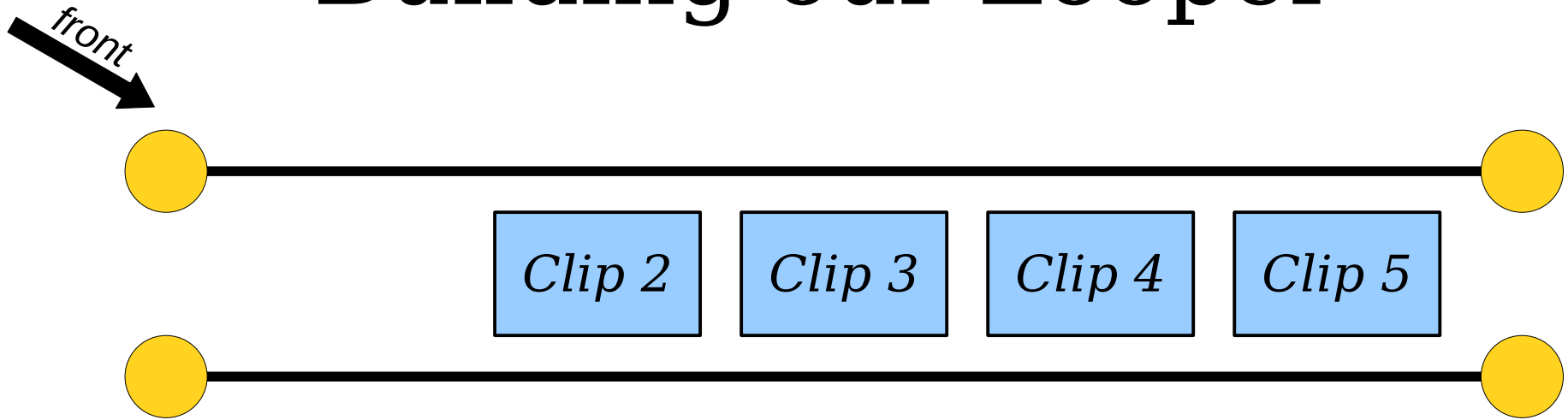
# Building our Looper



```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
  
}
```



# Building our Looper

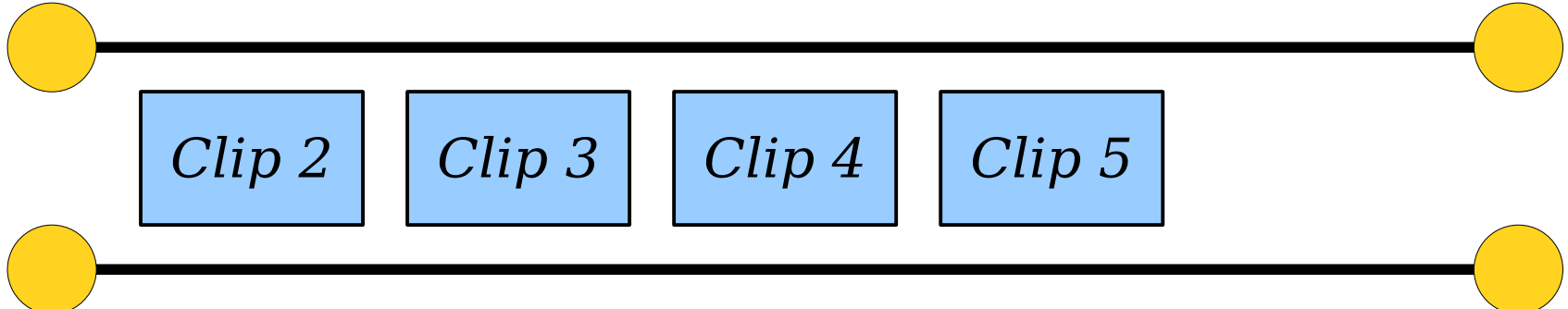


*Clip 1*

```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
  
}
```

# Building our Looper

front



The diagram shows a double-ended queue represented by two horizontal black lines. The top line has a yellow circle at its left end and another at its right end. The bottom line also has a yellow circle at its left end and another at its right end. Between these two lines, four light blue rectangular boxes are arranged horizontally, labeled 'Clip 2', 'Clip 3', 'Clip 4', and 'Clip 5' from left to right. An arrow labeled 'front' points towards the left end of the top line.

*Clip 1*

```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
  
}
```

# Building our Looper

front



*Clip 2*

*Clip 3*

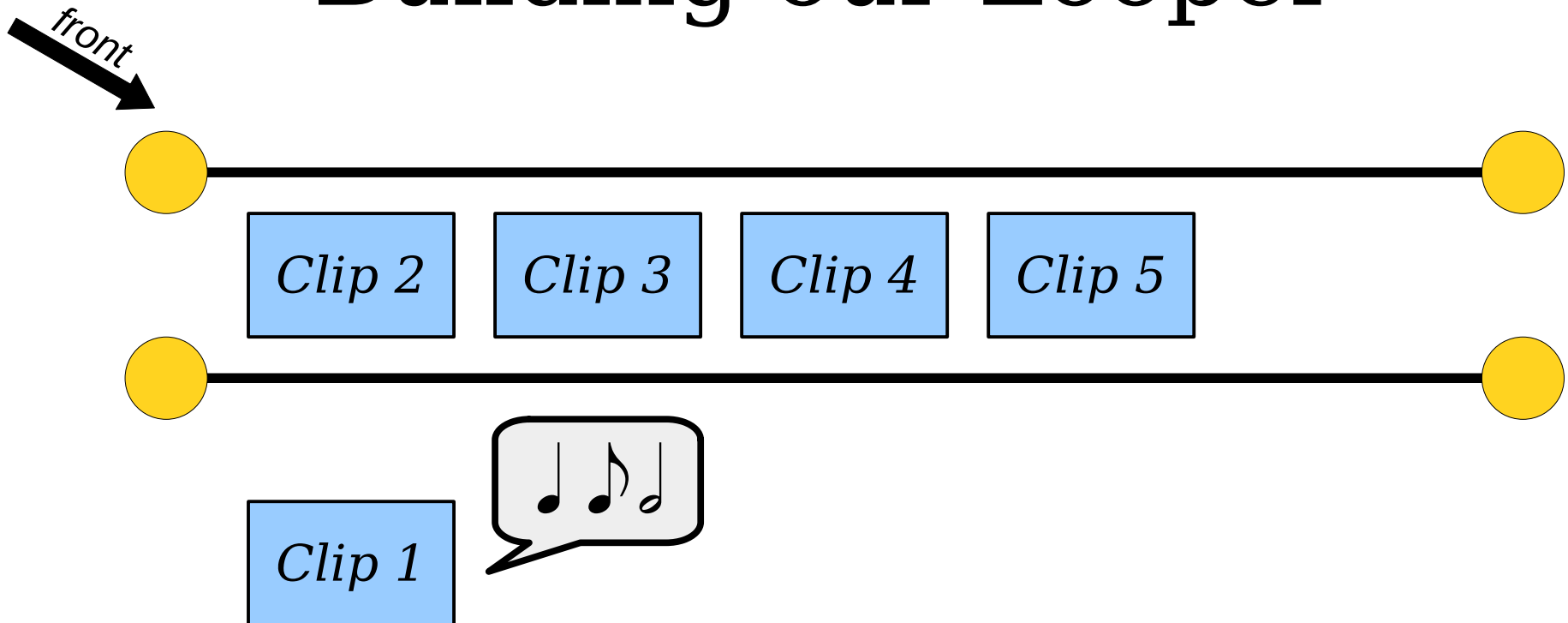
*Clip 4*

*Clip 5*

*Clip 1*

```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
}
```

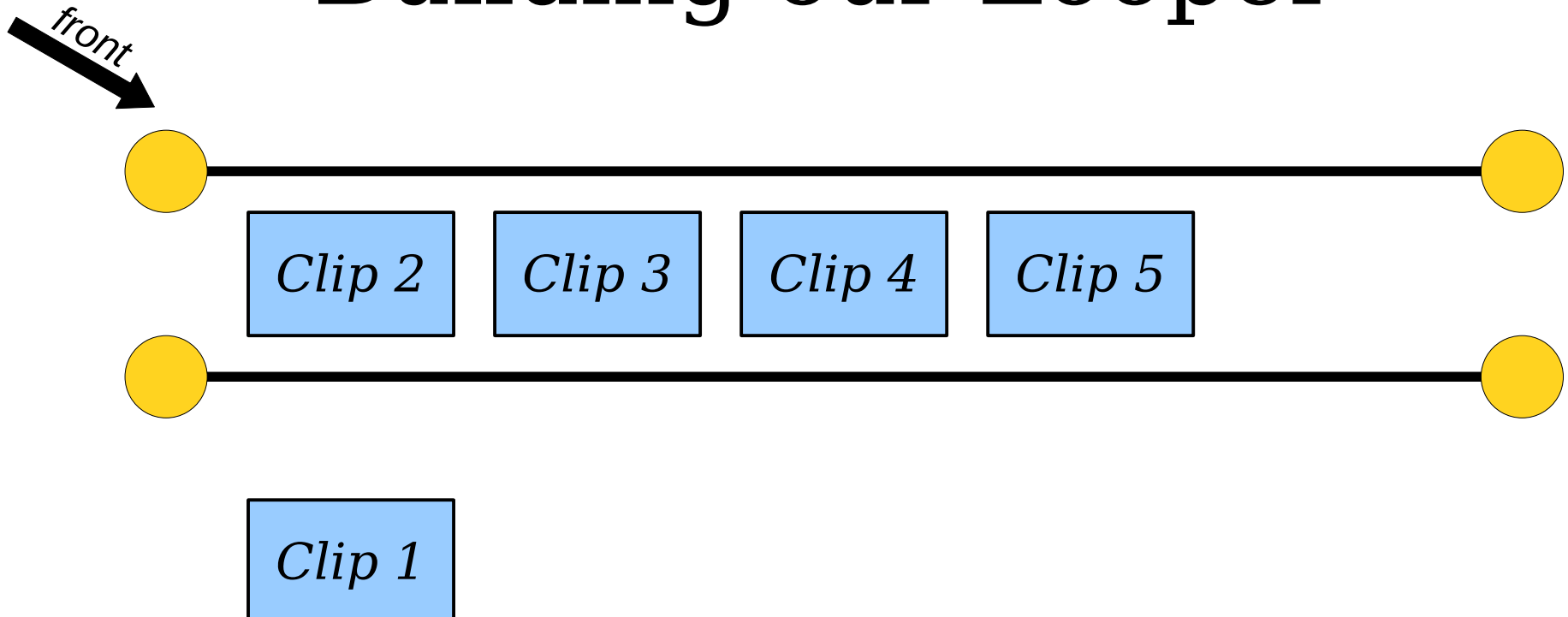
# Building our Looper



```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
}
```

# Building our Looper

front



*Clip 2*

*Clip 3*

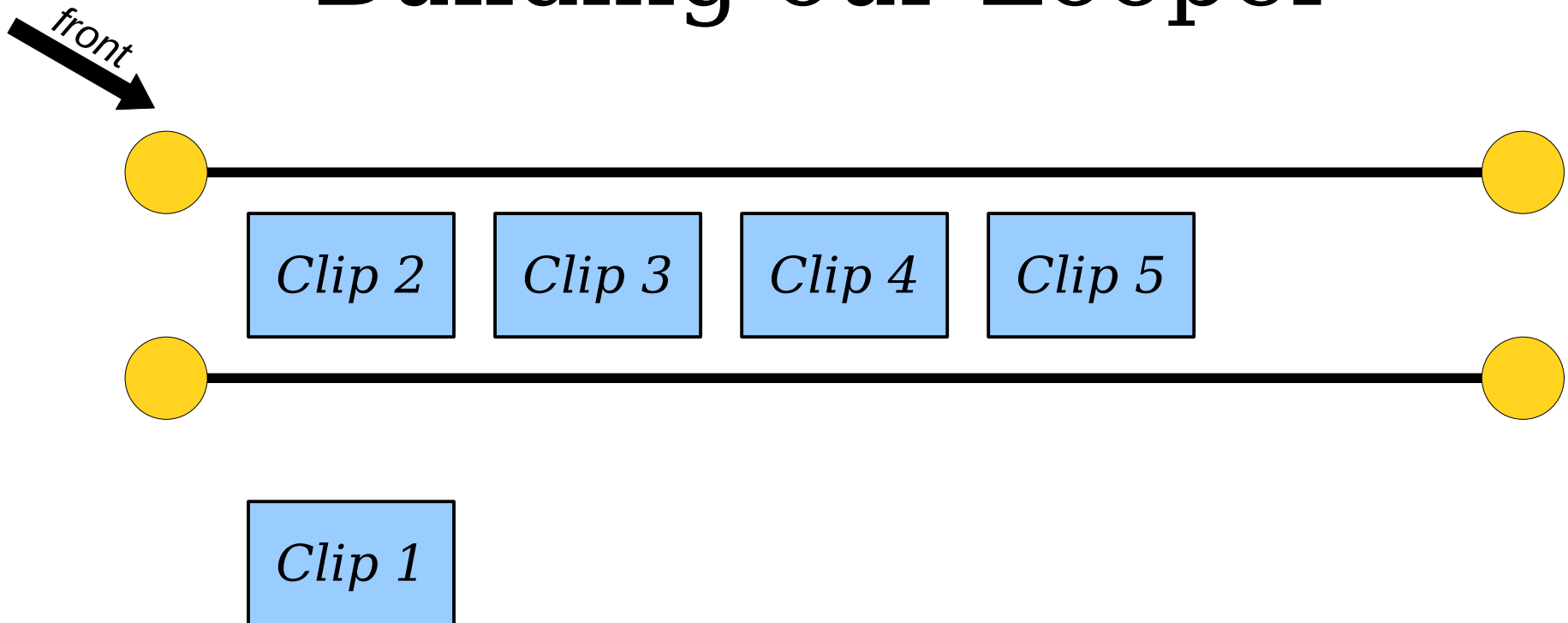
*Clip 4*

*Clip 5*

*Clip 1*

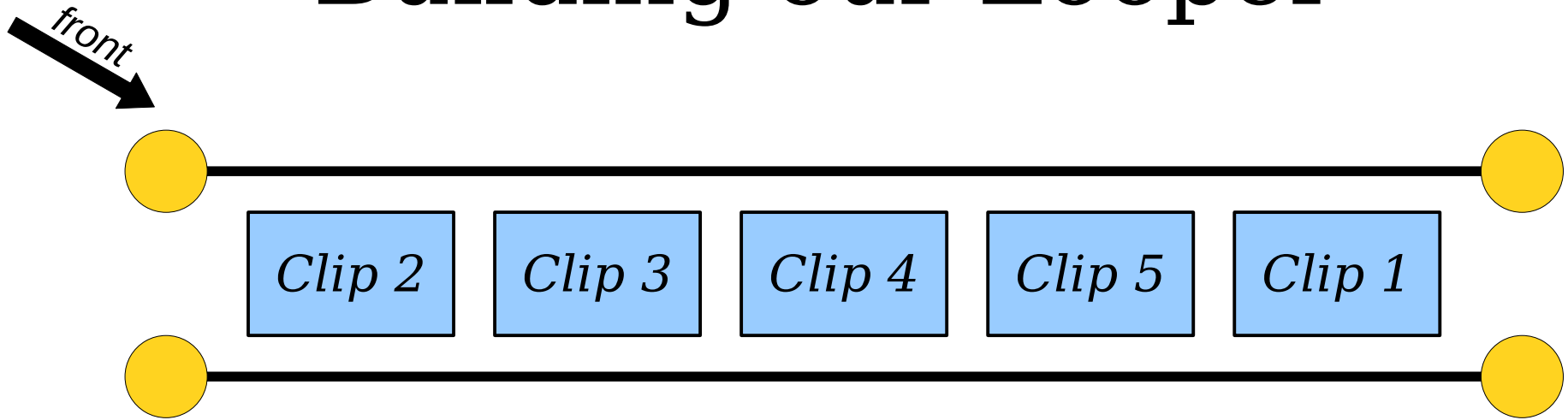
```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
}
```

# Building our Looper



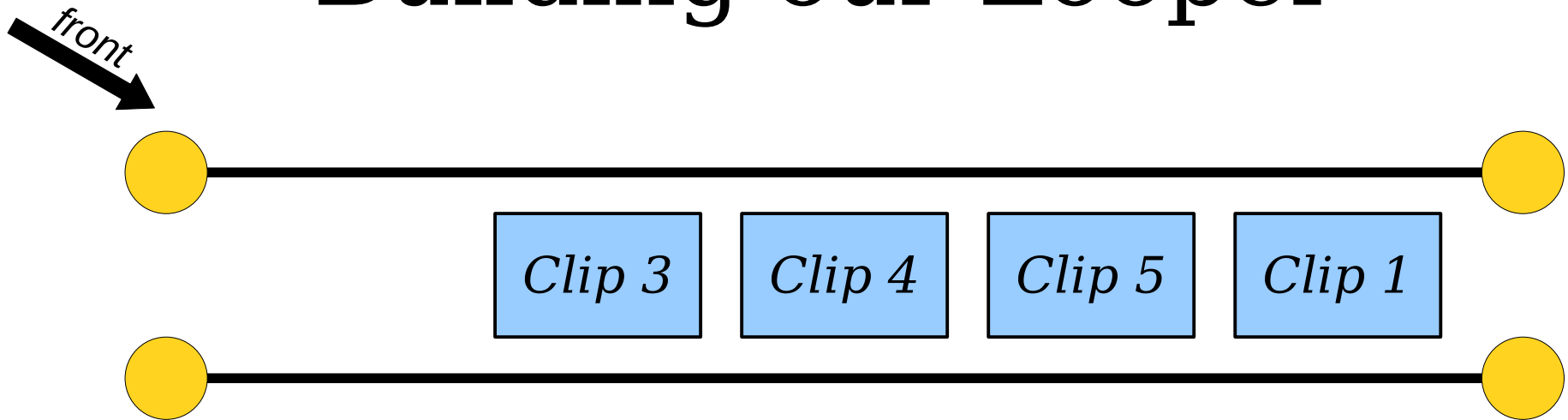
```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```

# Building our Looper



```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```

# Building our Looper



Clip 2

```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```



# Building our Looper

front

*Clip 3*

*Clip 4*

*Clip 5*

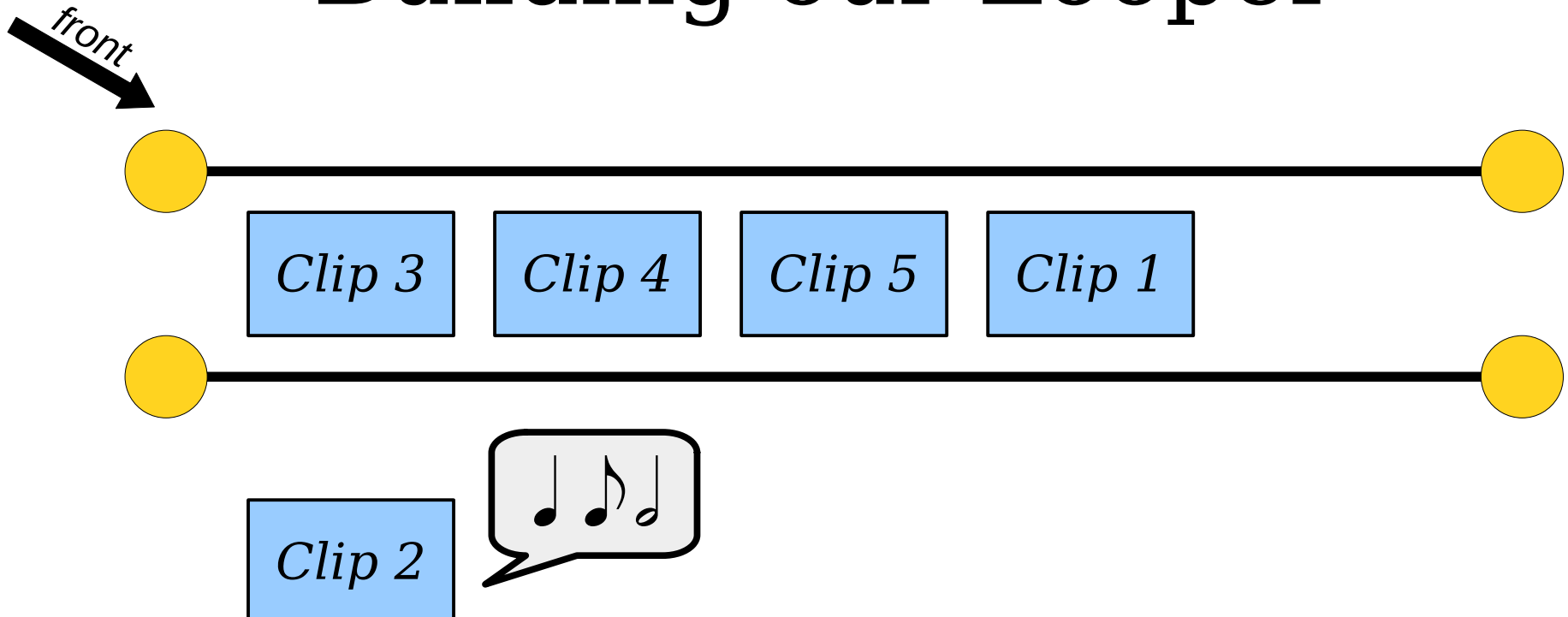
*Clip 1*

*Clip 2*

```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```

# Building our Looper

front



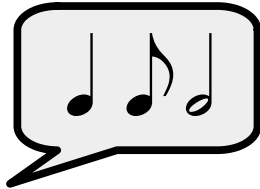
*Clip 3*

*Clip 4*

*Clip 5*

*Clip 1*

*Clip 2*



```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```

# Building our Looper

front

*Clip 3*

*Clip 4*

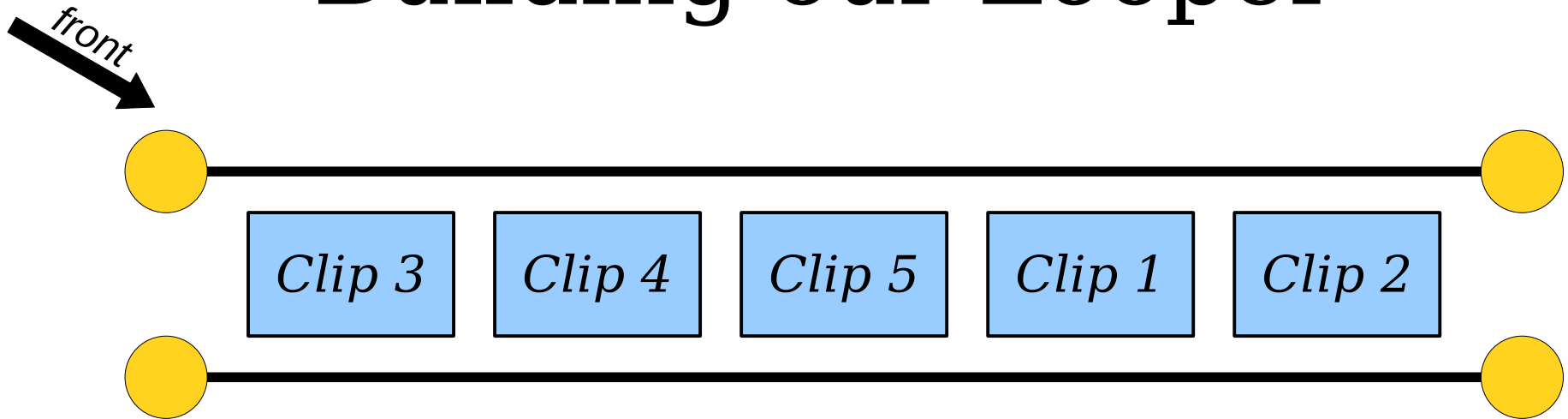
*Clip 5*

*Clip 1*

*Clip 2*

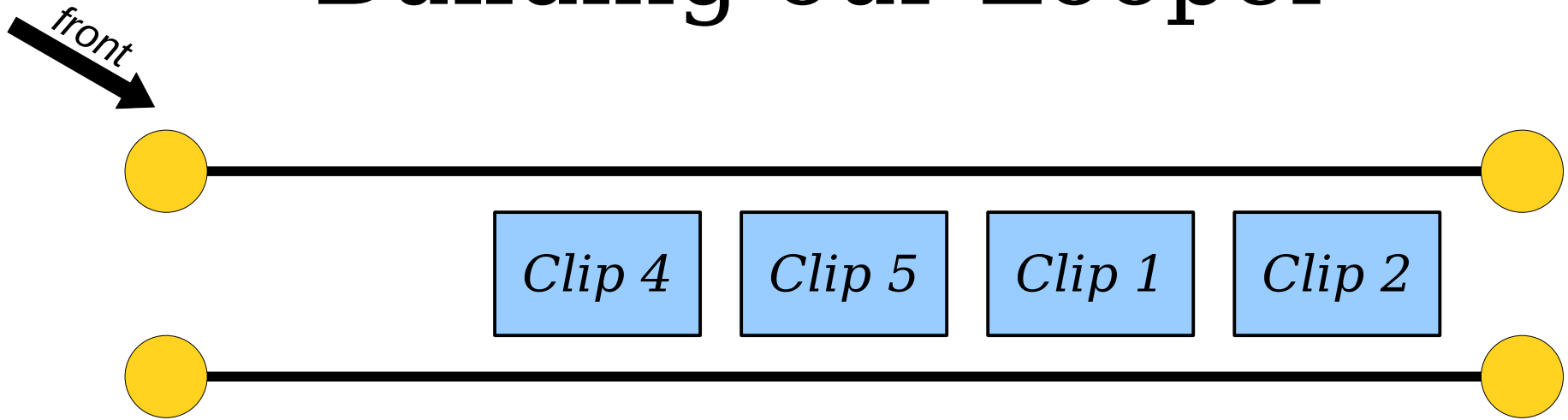
```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```

# Building our Looper



```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```

# Building our Looper



Clip 3

```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```

# Building our Looper

front



*Clip 4*

*Clip 5*

*Clip 1*

*Clip 2*

*Clip 3*

```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```

# Building our Looper

front



*Clip 4*

*Clip 5*

*Clip 1*

*Clip 2*

*Clip 3*



```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```

# Building our Looper

front

*Clip 4*

*Clip 5*

*Clip 1*

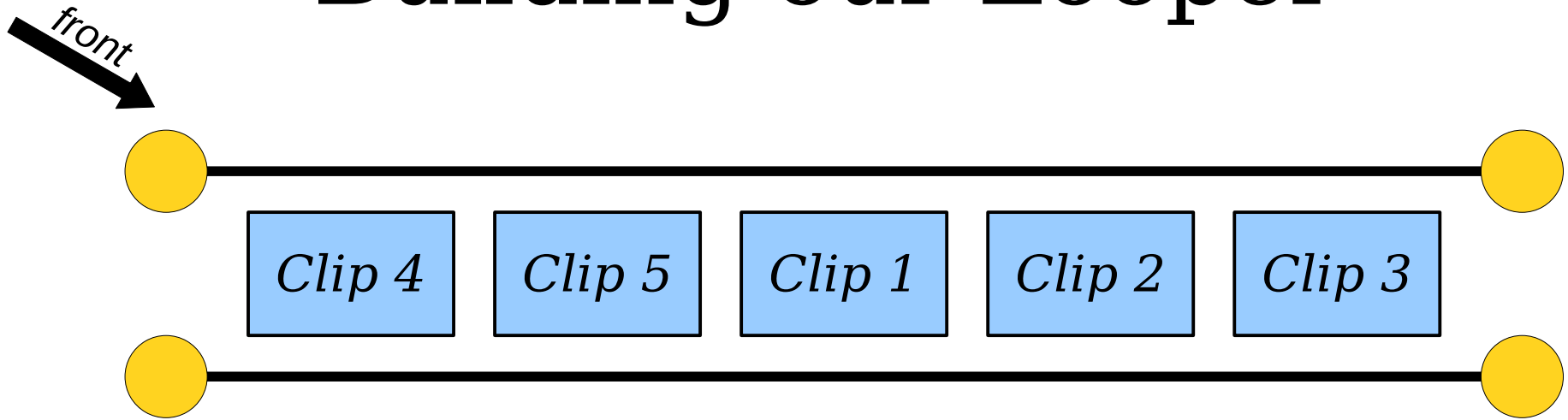
*Clip 2*

*Clip 3*

```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```



# Building our Looper



```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
    loop.enqueue(toPlay);  
}
```

# Enjoying Our Looper

Feeling musical? Want to contribute a loop for the next iteration of CS106B? Send me your .loop file and we'll add it to our collection!

# Changing our Looper

# Changing our Looper

```
Queue<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.dequeue();  
    playSound(toPlay.filename, toPlay.length);  
  
    loop.enqueue(toPlay);  
}
```

# Changing our Looper

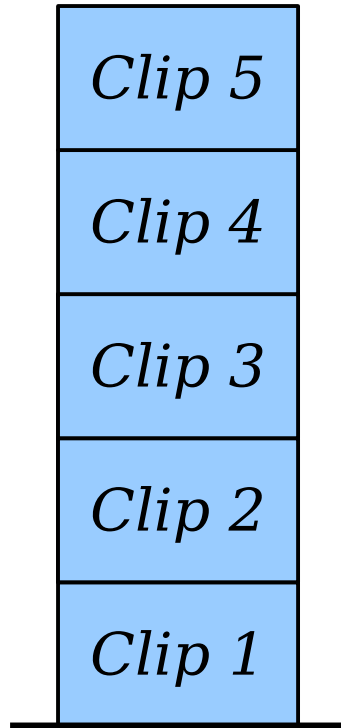
```
Stack<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.pop();  
    playSound(toPlay.filename, toPlay.length);  
  
    loop.push(toPlay);  
}
```

# Changing our Looper

Make a prediction – what are you going to hear when we use this version of the looper?

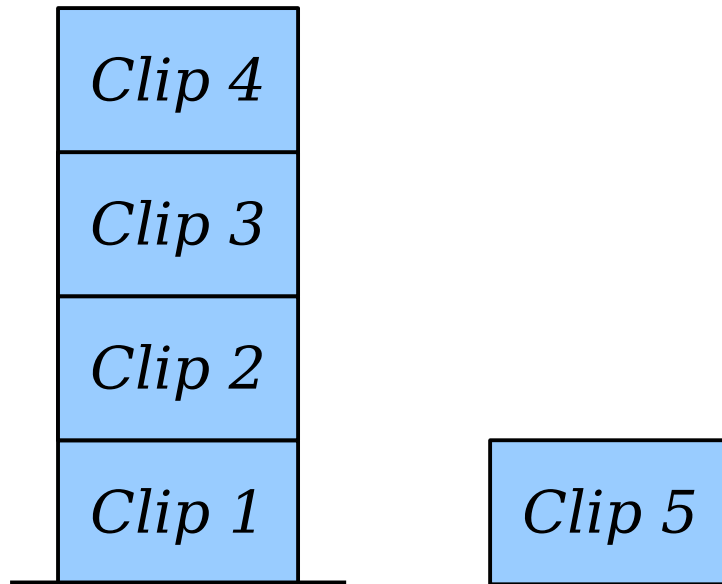
```
Stack<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.pop();  
    playSound(toPlay.filename, toPlay.length);  
    loop.push(toPlay);  
}
```

# Changing our Looper



```
Stack<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.pop();  
    playSound(toPlay.filename, toPlay.length);  
    loop.push(toPlay);  
}
```

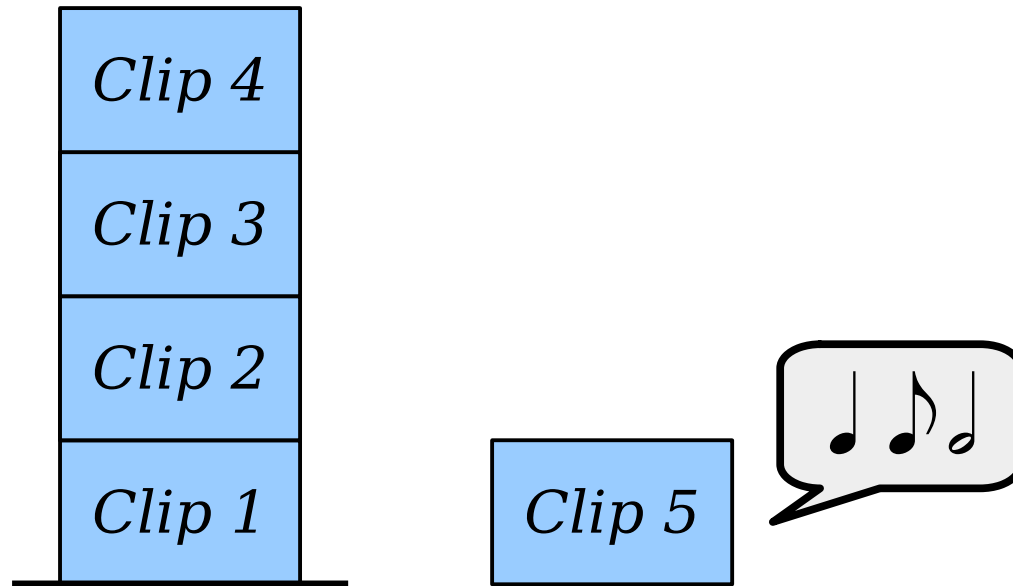
# Changing our Looper



```
Stack<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.pop();  
    playSound(toPlay.filename, toPlay.length);  
    loop.push(toPlay);  
}
```

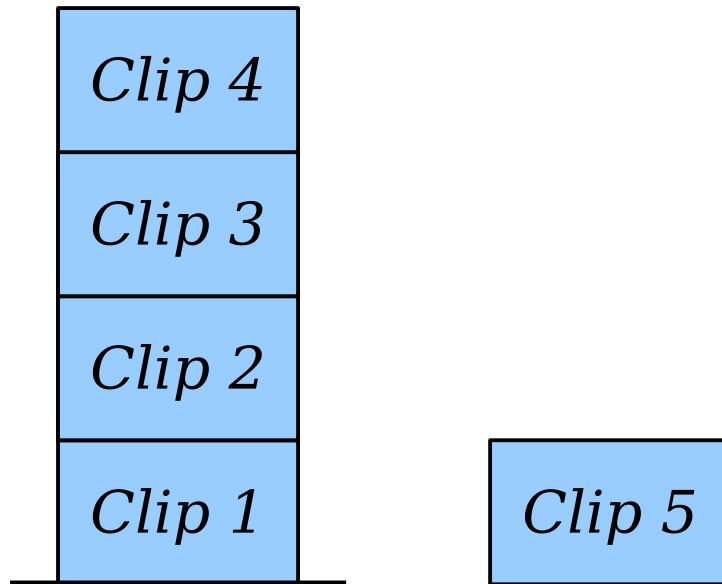


# Changing our Looper



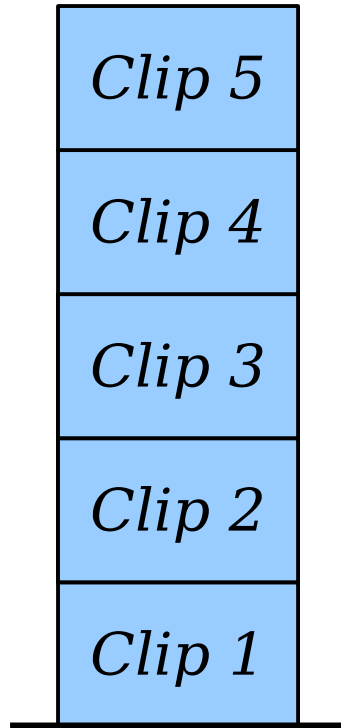
```
Stack<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.pop();  
    playSound(toPlay.filename, toPlay.length);  
    loop.push(toPlay);  
}
```

# Changing our Looper



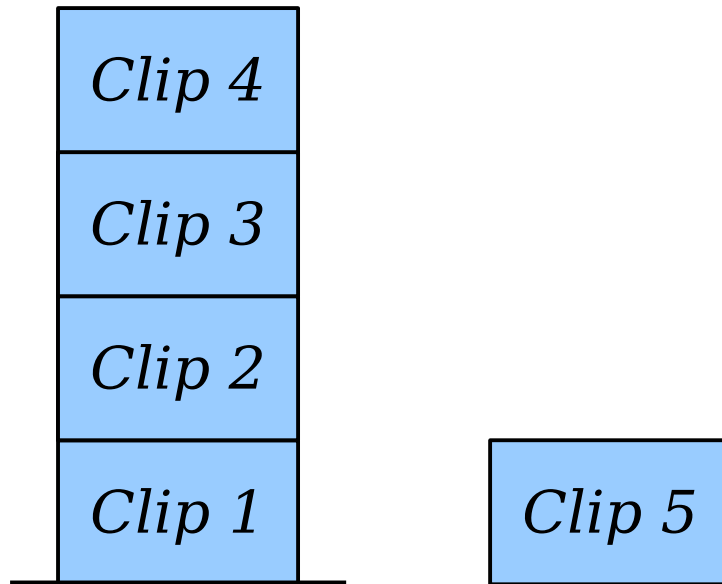
```
Stack<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.pop();  
    playSound(toPlay.filename, toPlay.length);  
    loop.push(toPlay);  
}
```

# Changing our Looper



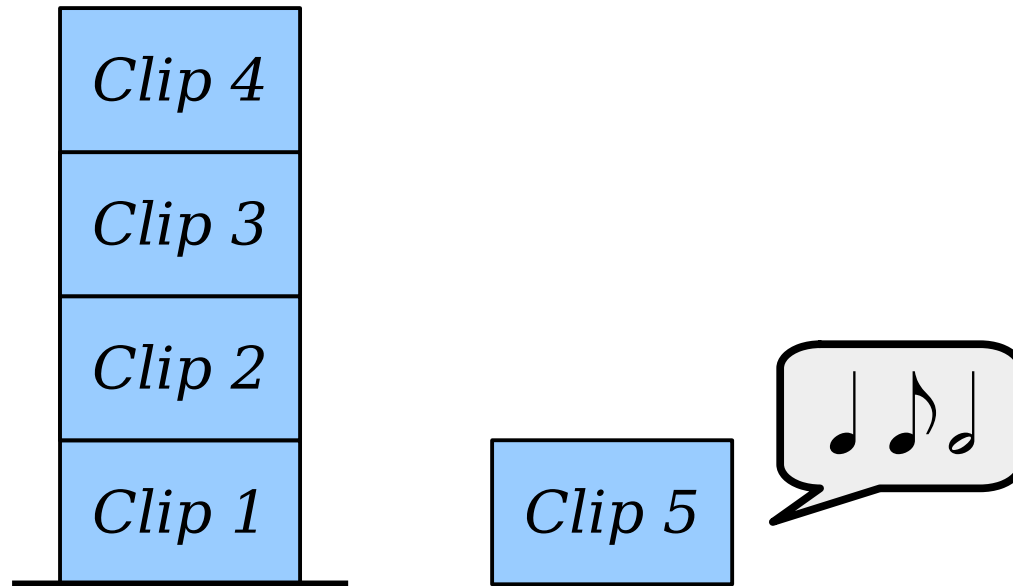
```
Stack<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.pop();  
    playSound(toPlay.filename, toPlay.length);  
    loop.push(toPlay);  
}
```

# Changing our Looper



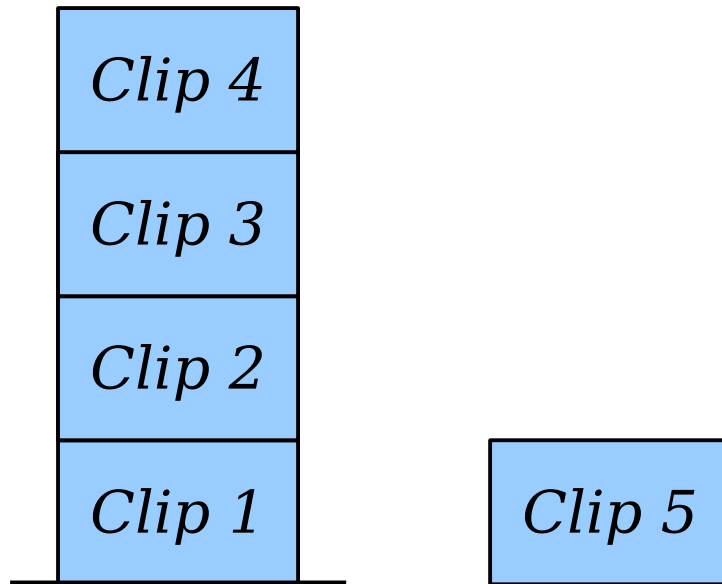
```
Stack<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.pop();  
    playSound(toPlay.filename, toPlay.length);  
    loop.push(toPlay);  
}
```

# Changing our Looper



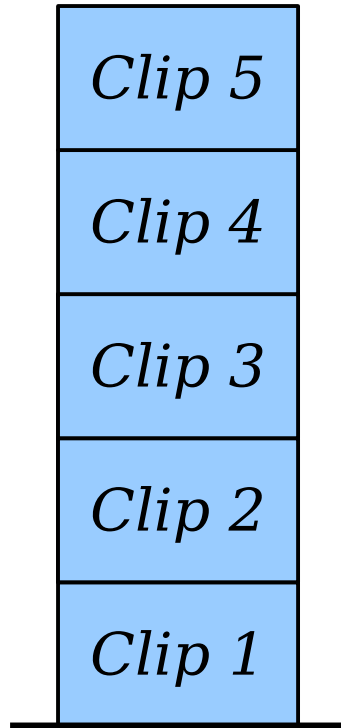
```
Stack<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.pop();  
    playSound(toPlay.filename, toPlay.length);  
    loop.push(toPlay);  
}
```

# Changing our Looper



```
Stack<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.pop();  
    playSound(toPlay.filename, toPlay.length);  
    loop.push(toPlay);  
}
```

# Changing our Looper



```
Stack<SoundClip> loop = loadLoop(/* ... */);  
while (true) {  
    SoundClip toPlay = loop.pop();  
    playSound(toPlay.filename, toPlay.length);  
    loop.push(toPlay);  
}
```

# Your Action Items

- ***Read Chapter 5.2 and 5.3.***
  - These sections cover more about the Stack and Queue type, and they're great resources to check out.
- ***Attend your first section!***
  - How exciting!
- ***Finish Assignment 1.***
  - Read the style guide up on the course website for more information about good programming style.
  - Review the Assignment Submission Checklist to make sure your code is ready to submit.



# Next Time

- ***Associative Containers***
  - Data sets aren't always linear!
- ***Maps and Sets***
  - Two ways to organize information.