Class Design and Const Correctness

Game Plan

- class design
- overview of A3
- implementing vector + gap-buffer
- const correctness

class design

review of class design

- **interface**: specifies the operations that can be performed on instances of the class.
- **implementation**: specifying how those operations are to be performed.

class design terminology

public members are basically methods, vars are excluded in public members

- **public members:** methods that are part of the interface, can be called by a client (and inside the class).

- **private members**: helper methods and variables that can only be accessed inside the class.

interface: usually placed in the .h file

```
class IntVector {
public:
    IntVector();
    IntVector(size_t count, int val = 0);
    void push_back(int element);
    void insert(size_t index, int element);
    int& at(size_t index);
    void reserve(size_t new_capacity);
    size_t size();
private:
    int* _elems;
    size_t _buffer_size, _actual_size;
}
```

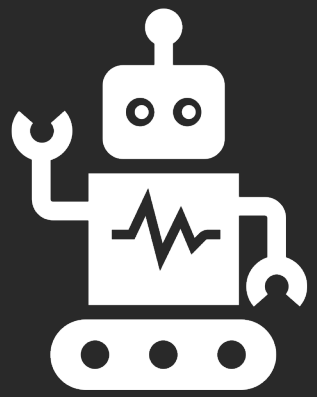
implementation: usually placed in the .cpp file

```
IntVector::IntVector(size_t count, int val) {
    _elems = new int[2*count];
    _buffer_size = 2*count;
    _logical_size = count;
    std::fill(_elems, _elems + count, val);
}

IntVector::push_back(int element) {
    if (_buffer_size == size()) reserve(2*size());
    _elems[_logical_size++] = element;
}
```

caveat with template classes

- We will soon be implementing template classes.
- Template classes require putting the implementation in the .h file (reasons we'll discuss later).
- To make it easier for you, we'll implement everything in the .h file now.



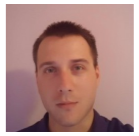
Example

overview of A3

Text Editor Wars

SIMPLE PROGRAMMER

PRODUCTS | ARTICLES | FREE COURSES | RESOURCES | JOBS | ABOUT



By Zlatin Stanimirovv
August 24, 2016

An Epic Tale of Comparison in The Text Editor Wars

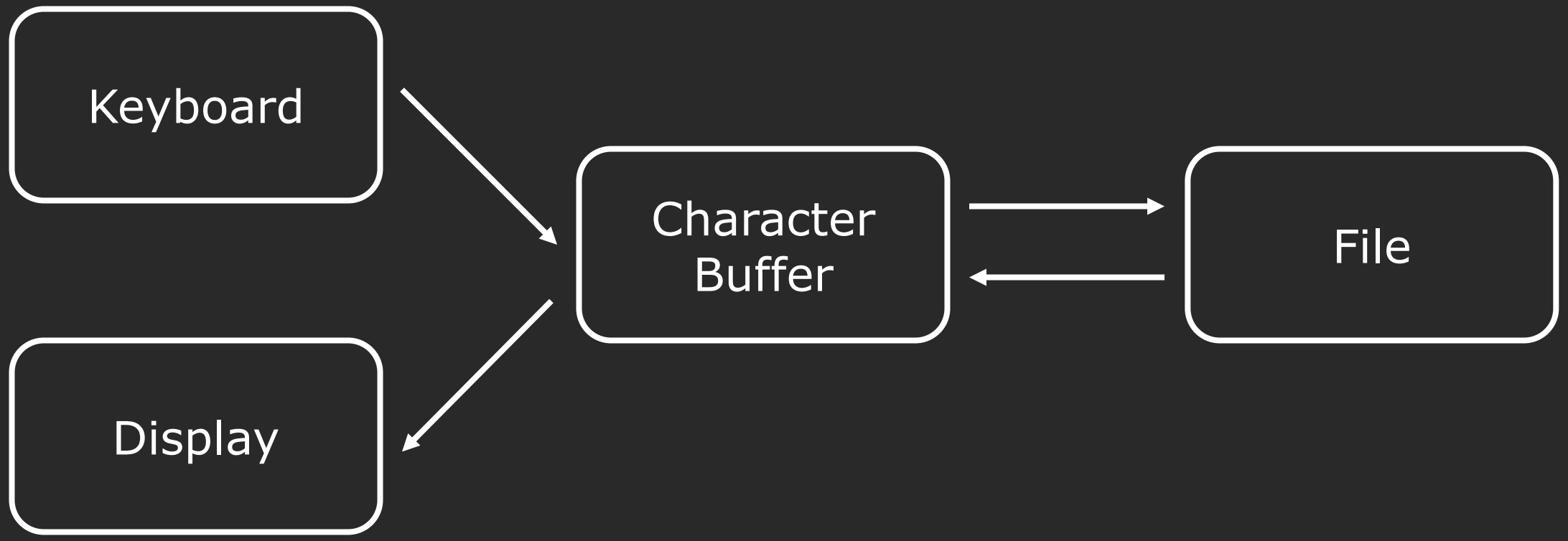
The programming community is filled with intellectuals who don't fight for decades about silly things, right?

WRONG!

Programmers fight about the silliest things I can imagine. We are talking about "my Pokemon is better than yours" type of shit here.

In order to understand The Text Editor Wars, we must time-travel into the past, to a place where computers were the size of buildings, network security was unthought of, and … ah, you get the idea.

basic functionality of a text editor



text editors need a character buffer

- user has a cursor, which moves around and allows the user to insert, remove or edit characters.
- easy approach: store the characters in a vector!

Problems with storing characters in a vector?

I	t	o	-	E	n		G	r	e	e	n		T	e	a									

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

	I	t	o	-	E	n		G	r	e	e	n		T	e	a								

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

6	I	t	o	-	E	n		G	r	e	e	n		T	e	a								

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

6		I	t	o	-	E	n		G	r	e	e	n		T	e	a							

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

6	-	I	t	o	-	E	n		G	r	e	e	n		T	e	a							

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

6	-		I	t	o	-	E	n		G	r	e	e	n		T	e	a						

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

6	-	p	I	t	o	-	E	n		G	r	e	e	n		T	e	a						

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

6	-	p	a	I	t	o	-	E	n		G	r	e	e	n		T	e	a					

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

6	-	p	a		I	t	o	-	E	n		G	r	e	e	n		T	e	a				

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

6	-	p	a	c	I	t	o	-	E	n		G	r	e	e	n		T	e	a				

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

6	-	p	a	c		I	t	o	-	E	n		G	r	e	e	n		T	e	a		

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

6	-	p	a	c	k	I	t	o	-	E	n		G	r	e	e	n		T	e	a			

Forgot to tell you…they came in a 6-pack!

Problems with storing characters in a vector?

6	-	p	a	c	k		I	t	o	-	E	n		G	r	e	e	n		T	e	a		

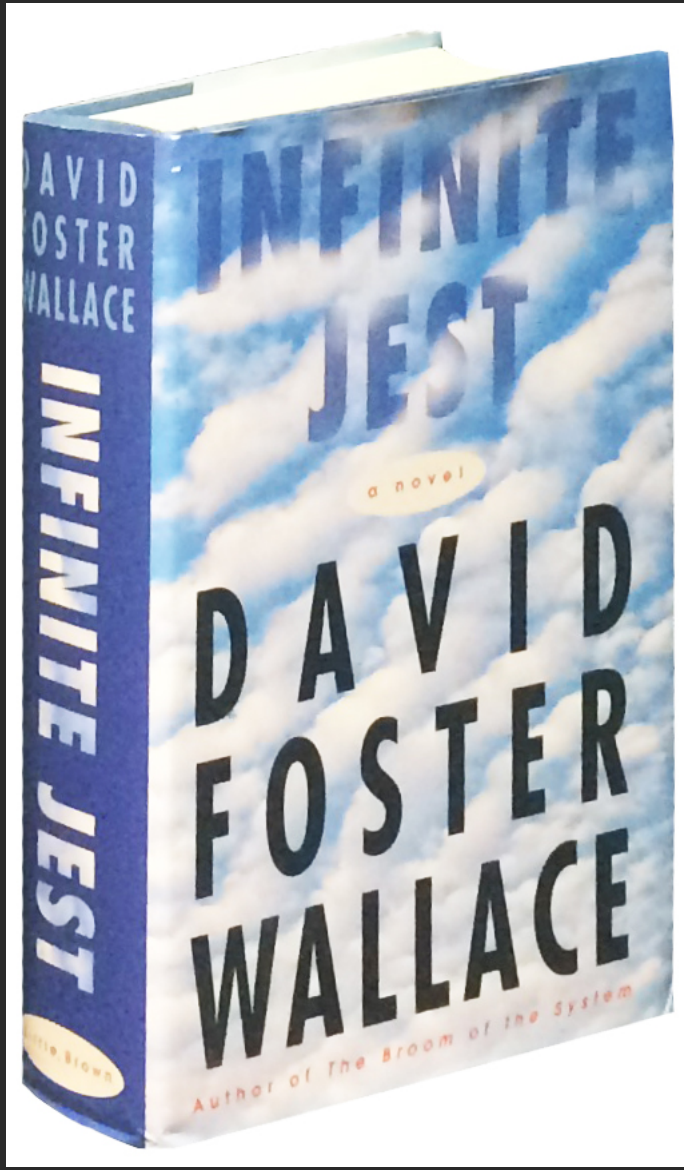
Forgot to tell you…they came in a 6-pack!

implementation: usually placed in the .cpp file

```
for (size_t i = 0; i < word.size(); ++i) {
    vec.insert(vec.begin(), word[i]);
}
```

Worse case: O(NL)
N = size of vector
L = size of word

543,709 words
2.68 million characters



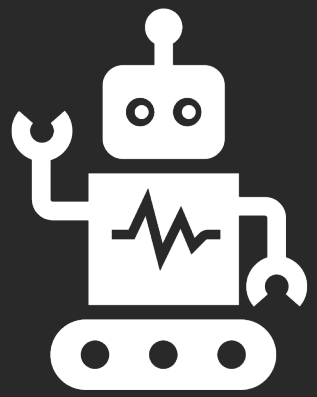
text editors must have fast insertion

"I am seated in an office, surrounded by heads and bodies."

(first sentence, ignoring forward)

text editors must have fast insertion

```
averyw09521@DN51udsd ~ % source /
Vector Creation Time: 249
GapBuffer Creation Time: 109
Vector Insertion Time: 541
GapBuffer Insertion Time: 0
YEAR OF GLAD
```



Example

Intuition: we only ever insert at the position of the cursor.

Can we make insert/delete fast?

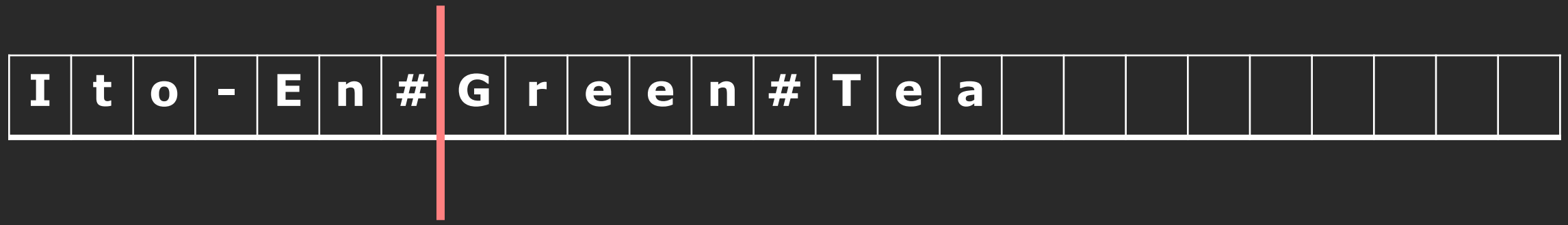
- Move cursor: O(1) – we keep track of a cursor index
- Insert: O(N) – we insert at index of the cursor

Can we make insert/delete fast?

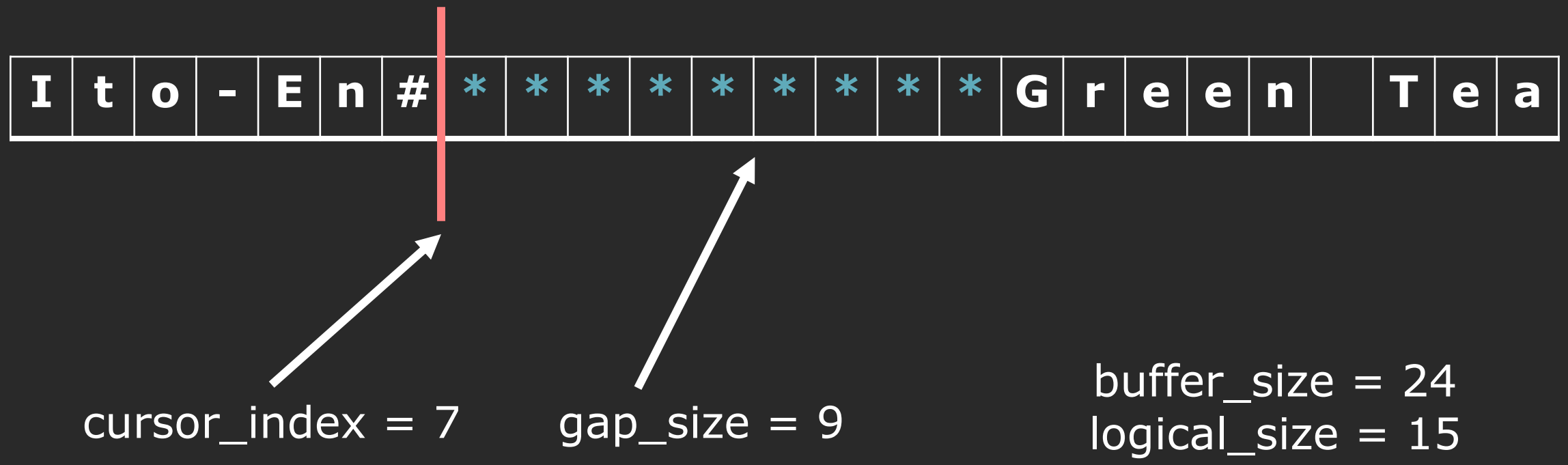
- Move cursor: somewhat frequent, jumps very rare
- Insert: very frequent in text editors

Gap Buffers

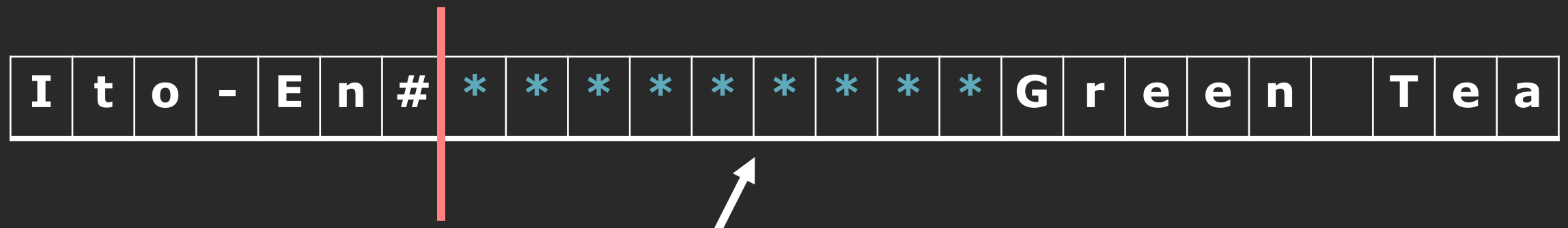
Intuition: leave a gap after the cursor to allow fast insertion.



Intuition: leave a gap after the cursor to allow fast insertion.



Insertion: write into the gap, move cursor forward

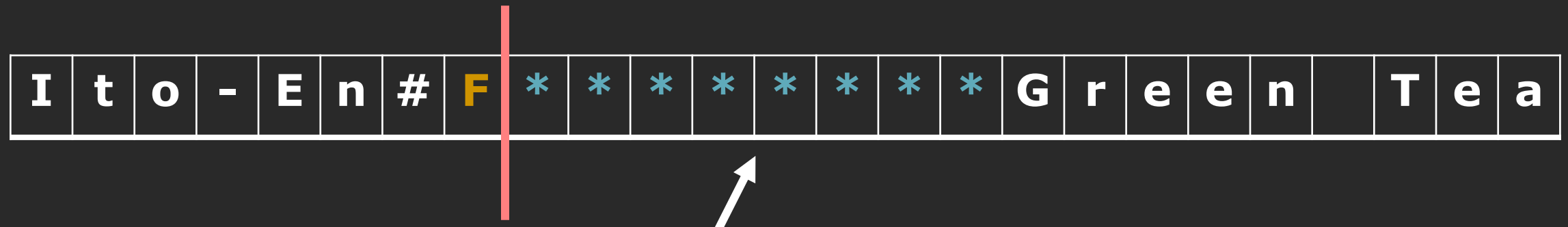


cursor_index = 7

gap_size = 9

buffer_size = 24
logical_size = 15

Insertion: write into the gap, move cursor forward

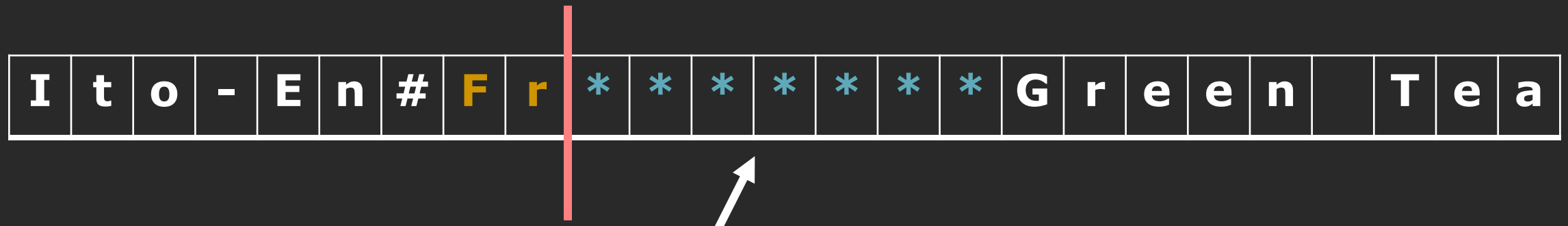


cursor_index = 8

gap_size = 8

buffer_size = 24
logical_size = 16

Insertion: write into the gap, move cursor forward

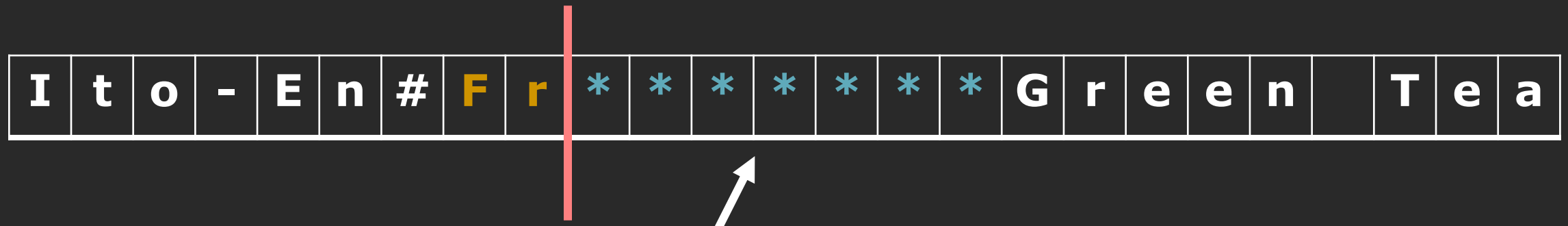


cursor_index = 9

gap_size = 7

buffer_size = 24
logical_size = 17

Deletion: expand gap, move cursor backward

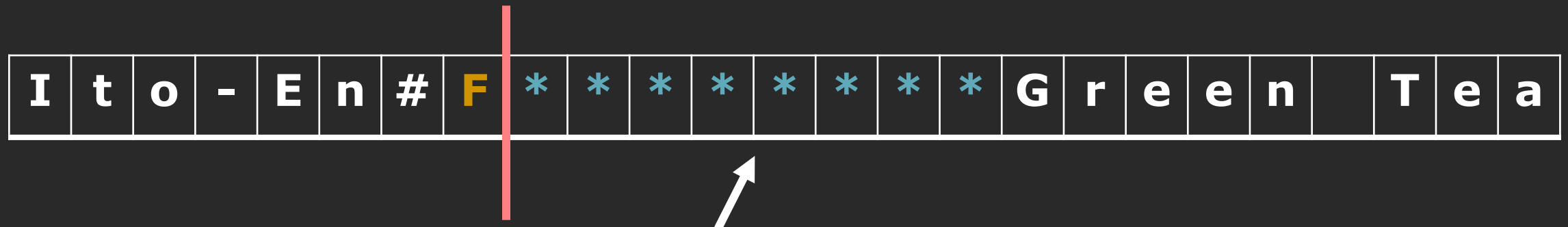


cursor_index = 9

gap_size = 7

buffer_size = 24
logical_size = 17

Deletion: expand gap, move cursor backward

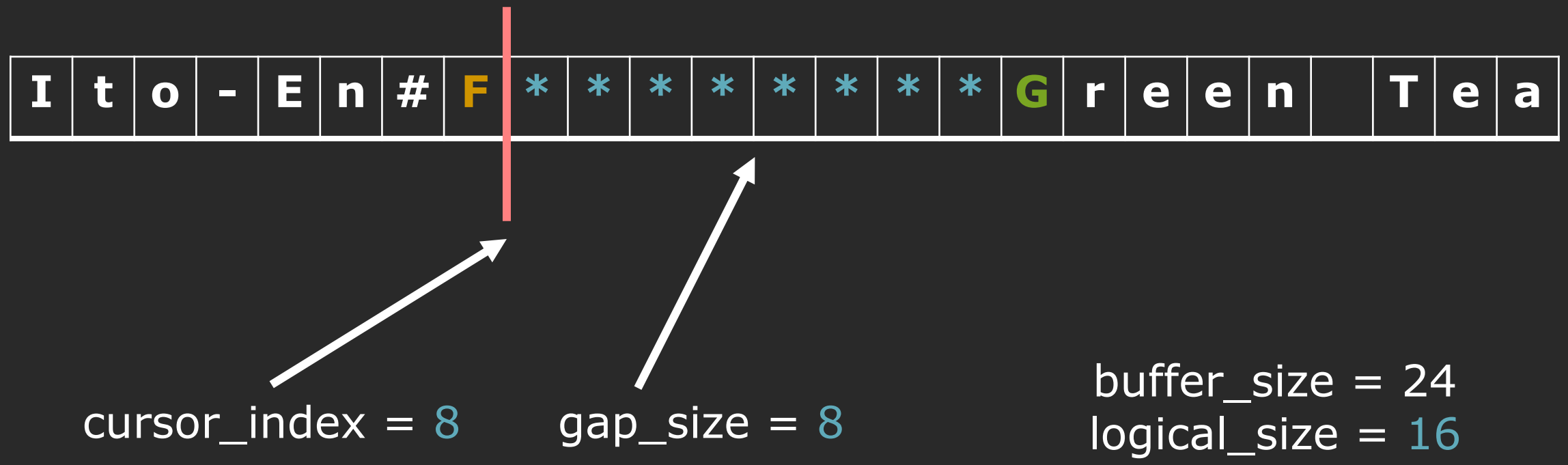


cursor_index = 8

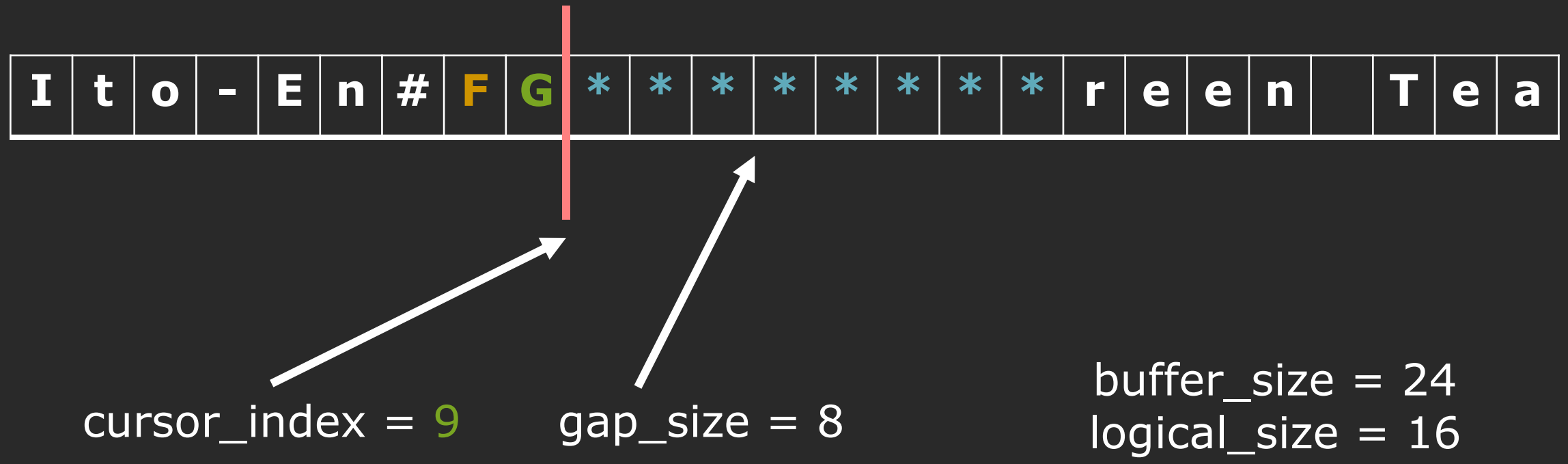
gap_size = 8

buffer_size = 24
logical_size = 16

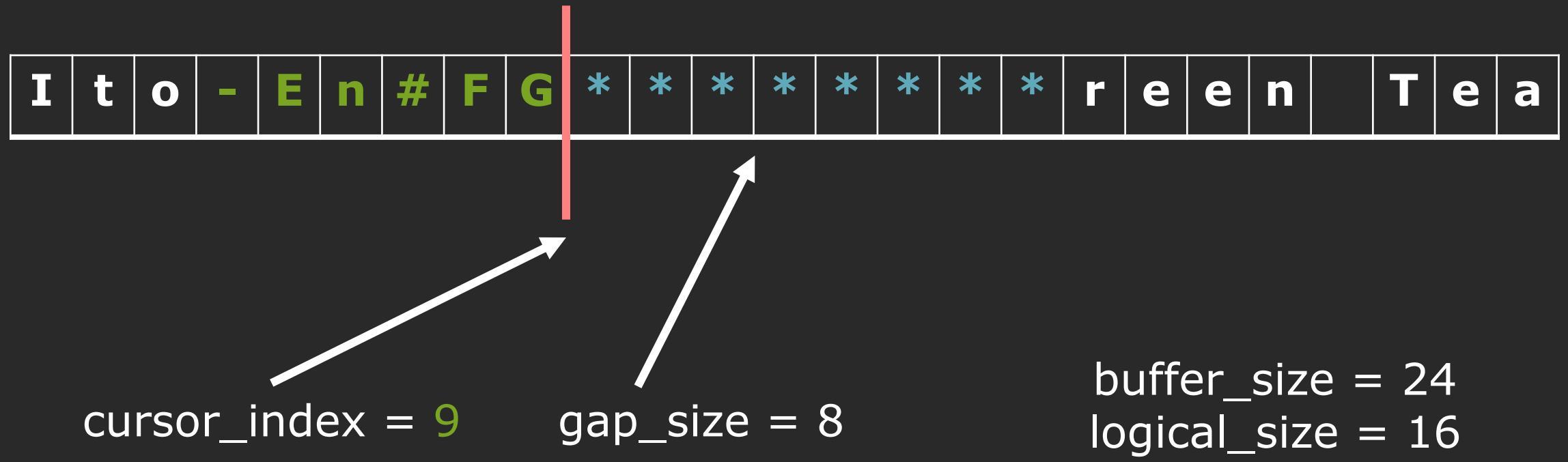
Move cursor: move the next character across the buffer.



Move cursor: move the next character across the buffer.



You can also move by an arbitrary amount (jump)



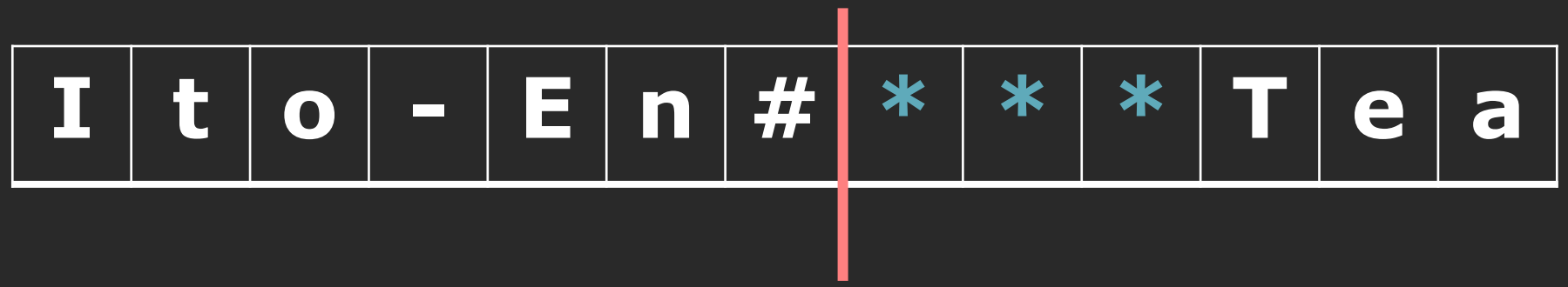
You can also move by an arbitrary amount (jump)

cursor_index = 3

gap_size = 8

buffer_size = 24
logical_size = 16

Another example



cursor_index = 7
gap_size = 3
buffer_size = 13
logical_size= 10

Another example

I	t	o	-	E	n	#	G	*	*	T	e	a

cursor_index = 8
gap_size = 2
buffer_size = 13
logical_size= 11

Another example

I	t	o	-	E	n	#	G	r	*	T	e	a

cursor_index = 9
gap_size = 1
buffer_size = 13
logical_size= 12

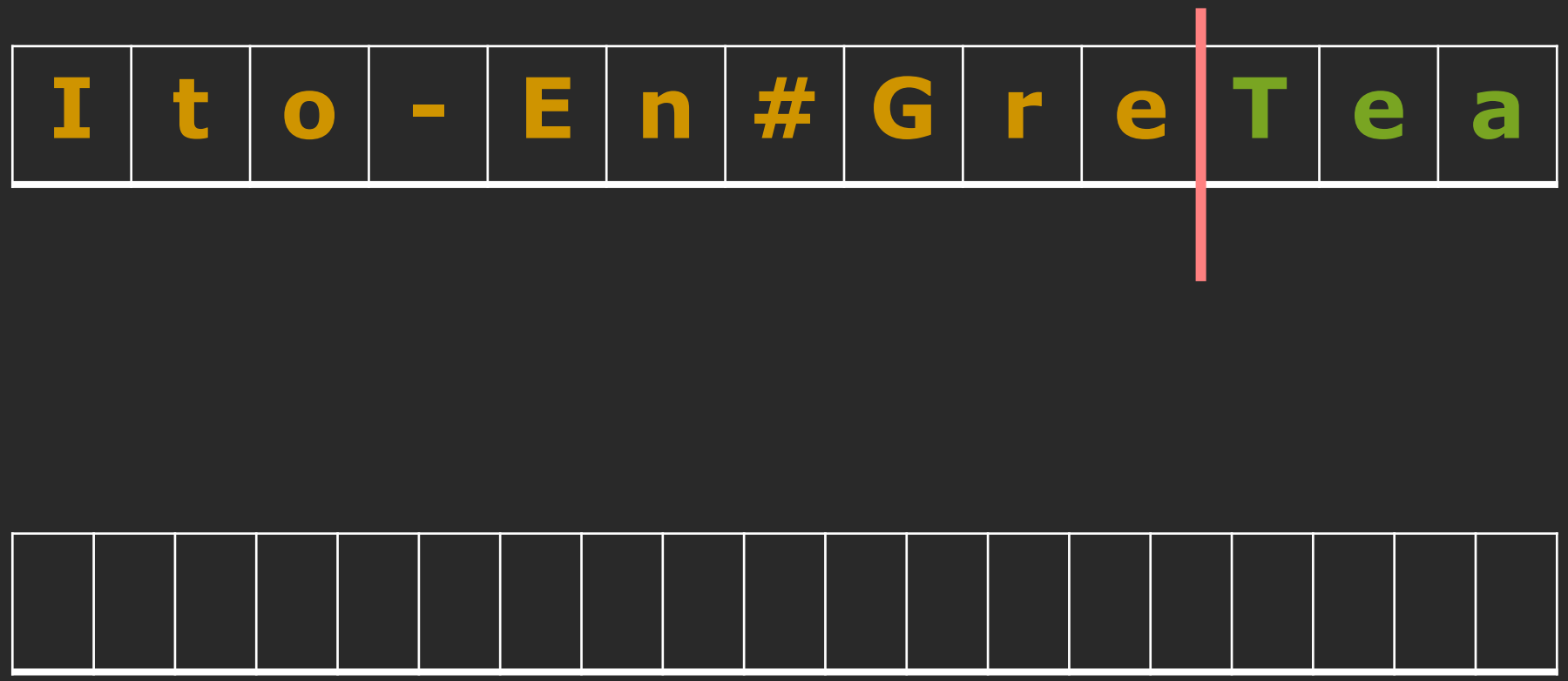
Another example

cursor_index = 10
gap_size = **0**
buffer_size = 13
logical_size= 13

What do we do now?

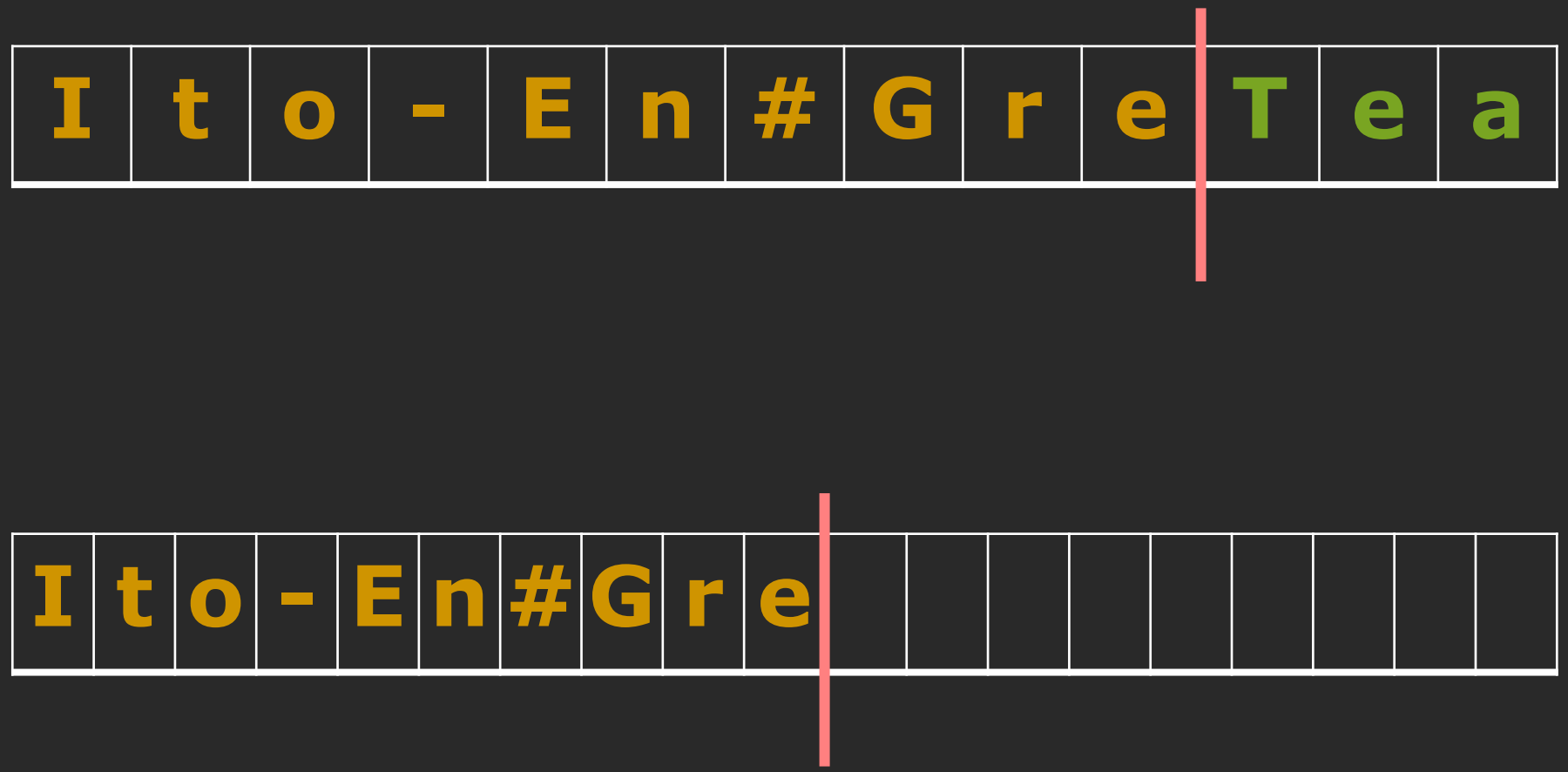
cursor_index = 10
gap_size = **0**
buffer_size = 13
logical_size= 13

Reserve: resize the buffer to a larger array.



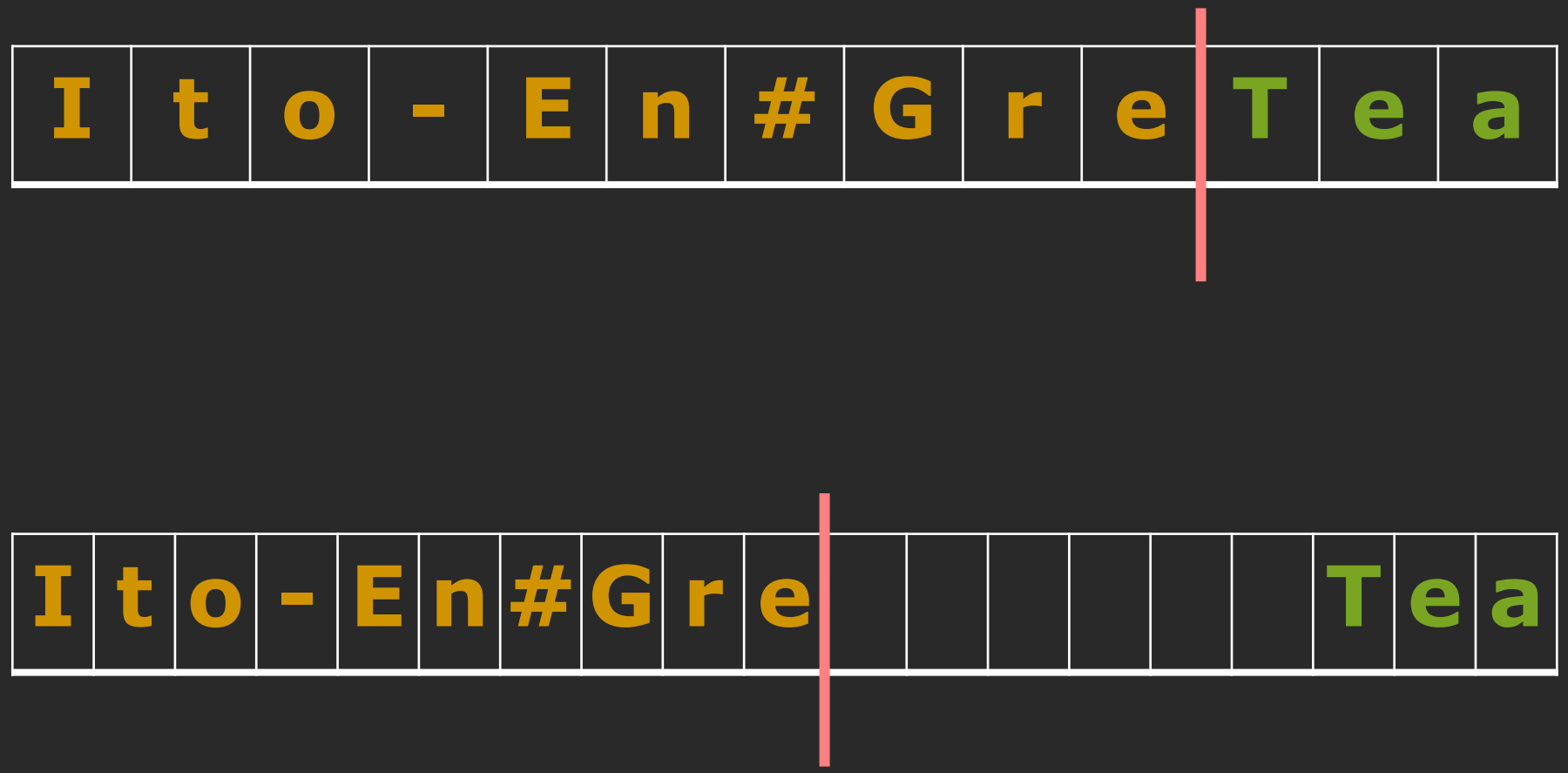
cursor_index = 10
gap_size = **0**
buffer_size = 13
logical_size= 13

Reserve: resize the buffer to a larger array.



cursor_index = 10
gap_size = **0**
buffer_size = 13
logical_size= 13

Reserve: resize the buffer to a larger array.



cursor_index = 10
gap_size = **0**
buffer_size = 13
logical_size= 13

Reserve: resize the buffer to a larger array.

cursor_index = 10
gap_size = **0**
buffer_size = 13
logical_size= 13

Reserve: resize the buffer to a larger array.

cursor_index = 10
gap_size = **0**
buffer_size = 13
logical_size= 13

cursor_index = 10
gap_size = **6**
buffer_size = 19
logical_size= 13

Reserve: resize the buffer to a larger array.

cursor_index = 10
gap_size = **0**
buffer_size = 13
logical_size= 13

cursor_index = 11
gap_size = **5**
buffer_size = 19
logical_size= 14

summary of GapBuffer interface

GapBuffer constructors

```
GapBuffer();
GapBuffer(size_type count,
          value_type val = value_type());
```

GapBuffer cursor operations

```
void insert_at_cursor(const_reference element);
void delete_at_cursor();
```

GapBuffer difficult cursor operations

```
void move_cursor(int delta);
void reserve(size_type new_size);
void debug();

size_type to_external_index(size_type array_index);
size_type to_array_index(size_type external_index);
```

GapBuffer retrieval operations

```
reference at(size_type index);
reference get_at_cursor();
```

GapBuffer const operations

```
size_type size() const;
size_type cursor_index() const;
bool empty() const;
```

pop quiz

what is the runtime of each of these operations?

Part 1: implement the following functions
(~28 lines of code)

```
GapBuffer();
GapBuffer(size_t count, char val = char());
void insert_at_cursor(const_reference element);
void delete_at_cursor();
reference at();
reference get_at_cursor();
size_type size() const;
size_type cursor_index() const;
bool empty() const;
```

const correctness

What is a const function?

A const function…

- cannot call other non-const member functions.
- cannot modify private members

What is a const instance?

A const instance (object)…

- cannot call non-const member functions.
- cannot modify private members (except in ctor & dtor)

Why might const instances be important?

```
int foo(const GapBuffer& buff) {
    buff.insert_at_cursor('c'); // BAD
    buff.size(); // OKAY

}
```

Interface for GapBuffer

```
GapBuffer();
GapBuffer(size_t count, char val = char());
void insert_at_cursor(const_reference element);
void delete_at_cursor();
reference at(size_type pos);
reference get_at_cursor();
size_type size() const;
size_type cursor_index() const;
bool empty() const;
```

Interface for const GapBuffer

```
GapBuffer();
GapBuffer(size_t count, char val = char());
void insert_at_cursor(const_reference element);
void delete_at_cursor();
reference at(size_type pos);
reference get_at_cursor();
size_type size() const;
size_type cursor_index() const;
bool empty() const;
```

Interface for const GapBuffer

```
GapBuffer();
GapBuffer(size_t count, char val = char());
void insert_at_cursor(const_reference element);
void delete_at_cursor();
reference at(size_type pos);
reference get_at_cursor();
size_type size() const;
size_type cursor_index() const;
bool empty() const;
```

what's a problem with this interface?

Members that return a reference should be usable for read-only access.

```
int foo(const GapBuffer& buff) {
    buff.at(3); // ERROR
    buff.get_at_cursor(); // ERROR

}
```

Interface for const GapBuffer

```
GapBuffer();
GapBuffer(size_t count, char val = char());
void insert_at_cursor(const_reference element);
void delete_at_cursor();
reference at(size_type pos) const;
reference get_at_cursor() const;
size_type size() const;
size_type cursor_index() const;
bool empty() const;
```

Can we just add a const?

Problem: a const member cannot return a non-const reference.

```
int foo(const GapBuffer& buff) {
    buff.at(3) = 3; // BAD!
    buff.get_at_cursor(); // OKAY

}
```

How do we allow both const and non-const uses?

Old Interface for const GapBuffer

```
reference at(size_type pos);

reference get_at_cursor();
```

Const-Correct Interface for const GapBuffer

```
reference at(size_type pos);
const_reference at(size_type pos) const;

reference get_at_cursor();
const_reference get_at_cursor() const;
```

compiler will choose the correct version

dependent qualified names

Not sure which class this is from?

It's from the GapBuffer class.

I know what this type is!

```
reference GapBuffer::at(size_type pos) {
    // implementation
}
```

Return value appears before class name.

dependent qualified names

Super explicitly, here is where reference is from.

```
typename GapBuffer::reference
                GapBuffer::at(size_type pos) {
    // implementation
}
```

Return value appears before class name.

static_cast/const_cast trick

```
typename GapBuffer::reference
                GapBuffer::at(size_type pos) {
     // exactly the same as below

}

typename GapBuffer::const_reference
                GapBuffer::at(size_type pos) const {
     // implementation
}
```

Can we unify the implementations?

static_cast/const_cast trick

```
typename GapBuffer::reference
                GapBuffer::at(size_type pos) {
     at(pos); // infinite recursion!

}

typename GapBuffer::const_reference
                GapBuffer::at(size_type pos) const {
     // implementation
}
```

"this" is non-const, so calls non-const at = recursion

static_cast/const_cast trick

```
typename GapBuffer::reference
                GapBuffer::at(size_type pos) {
     at(pos); // infinite recursion!

}
```

Let's do some surgery on "this" so we can call const version.

static_cast/const_cast trick

```
typename GapBuffer::reference
                GapBuffer::at(size_type pos) {
     this->at(pos);

}
```

Explicitly write out the “this” implicit parameter.

static_cast/const_cast trick

```
typename GapBuffer::reference
                GapBuffer::at(size_type pos) {
     static_cast<const GapBuffer*> (this)->at(pos);

}
```

Literally cast “this” to pointer to const instance, then call at.

static_cast/const_cast trick

```
typename GapBuffer::reference
                GapBuffer::at(size_type pos) {
     return static_cast<const GapBuffer*> (this)->at(pos);

}
```

Still not perfect yet…the call returns a const-reference.

static_cast/const_cast trick

```
typename GapBuffer::reference
                GapBuffer::at(size_type pos) {
     return const_cast<reference>(
                static_cast<const GapBuffer*> (this)->at(pos)
            );
}
```

Last step, convert the const reference to a non-const reference.

Next time

Operators