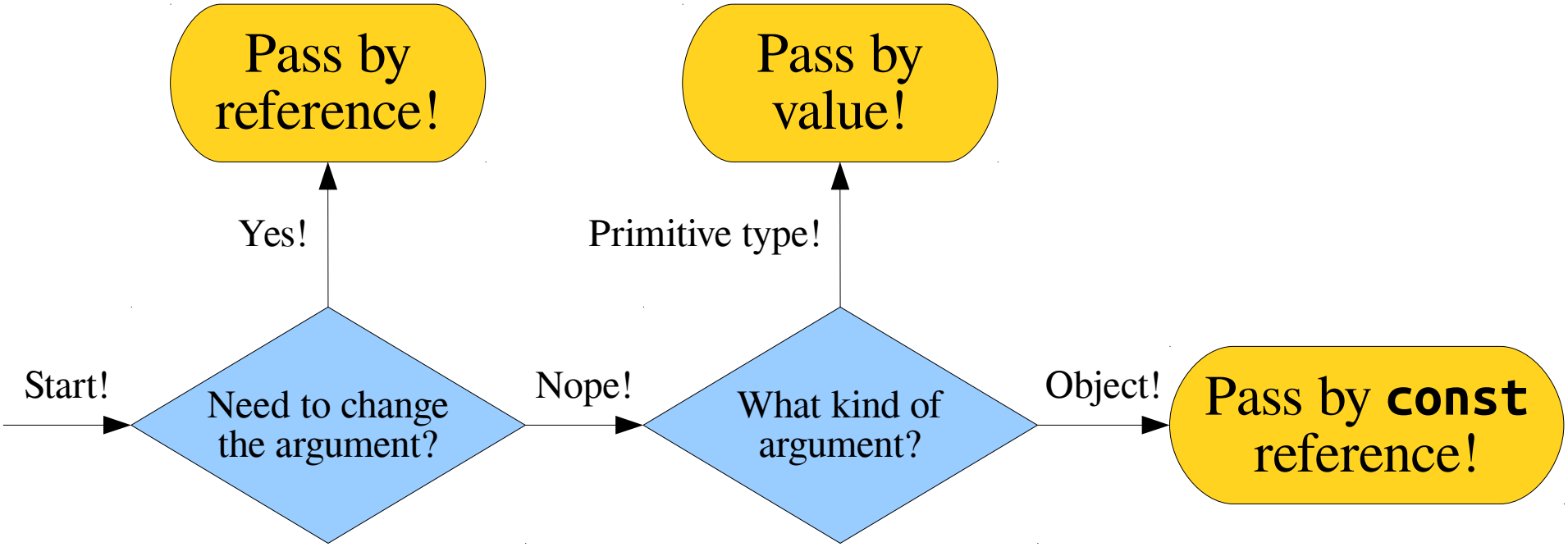
Collections, Part Three

Outline for Today

- ***Lexicon***
  - Storing a collection of words.
- ***HashSet***
  - Storing a group of whatever you'd like.
- ***HashMap***
  - A powerful, fundamental container.

Recap from Last Time

Parameter Flowchart



New Stuff!

Lexicon

Lexicon

- A **`Lexicon`** is a container that stores a collection of words.
- The `Lexicon` is designed to answer the following question efficiently:

Given a word, is it contained in the `Lexicon`?

- The Lexicon does *not* support access by index. You can't, for example, ask what the 137th English word is.
- However, it *does* support questions of the form "does this word exist?" or "do any words have this as a prefix?"

Tautonyms

- A ***tautonym*** is a word formed by repeating the same string twice.
  - For example: murmur, couscous, papa, etc.
- What English words are tautonyms?

Some Aa

http://upload.wikimedia.org/wikipedia/commons/f/f1/Aa_large.jpg

One Bulbul

More than One Caracara

http://www.greglasley.net/images/CO/Crested-Caracara-F3.jpg

Introducing the Dikdik

And a Music Recommendation

Time-Out for Announcements!

Assignment 2

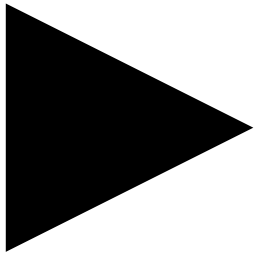
- Assignment 2 (Fun with Collections) goes out today. It's due next Friday.
  - Explore the impact of sea level rise.
  - Build a personality quiz!
- We've provided a suggested timetable for completing this assignment on the front page of the handout. Aim to stick to this timeline; you've got plenty of time to complete things if you start early.
- ***You must complete this assignment individually***. Working in pairs is not permitted on this assignment.

LaIR Closure

- The LaIR will be closed on Sunday in observance of Dr. Martin Luther King, Jr. Day.
- The LaIR will, however, be open on Monday during the usual 7PM – 11PM time slot.

CS Undergrad Research Panel

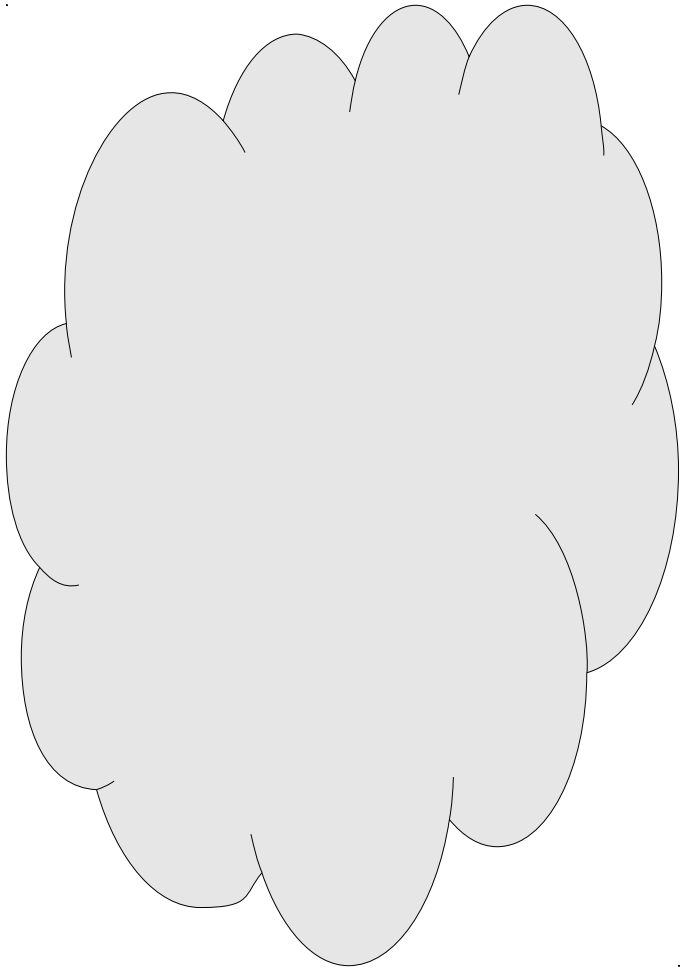
- The CS department is putting on an undergraduate research panel next Wednesday at 5:30PM in Gates 415.
- Curious to hear what it's like to do research in CS? Stop on by! An RSVP is requested using ***this link***.



HashSet

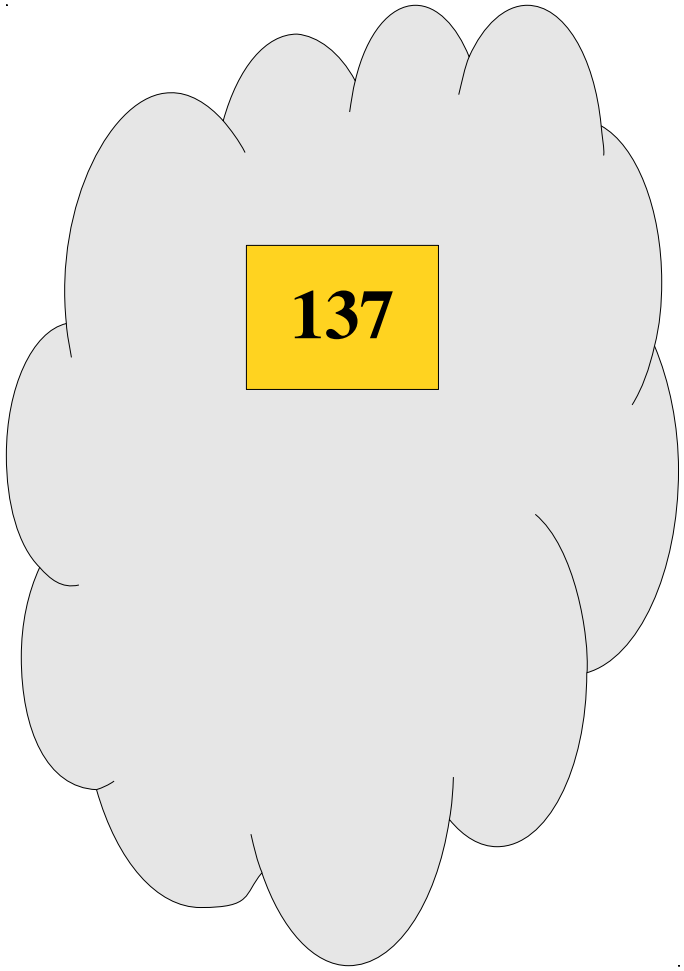
HashSet

- The **`HashSet`** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



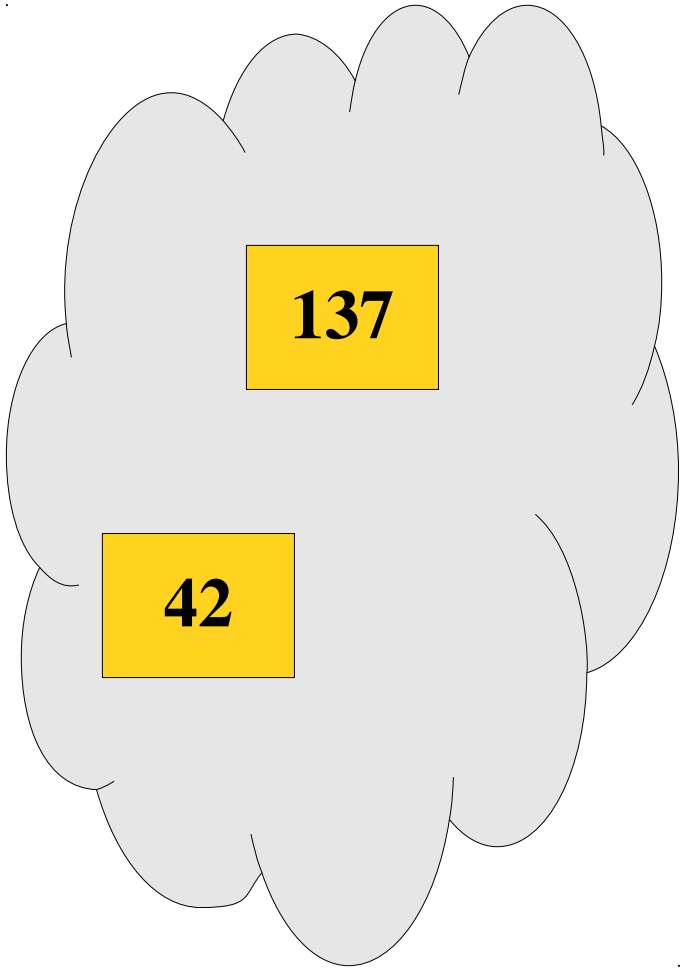
HashSet

- The **HashSet** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



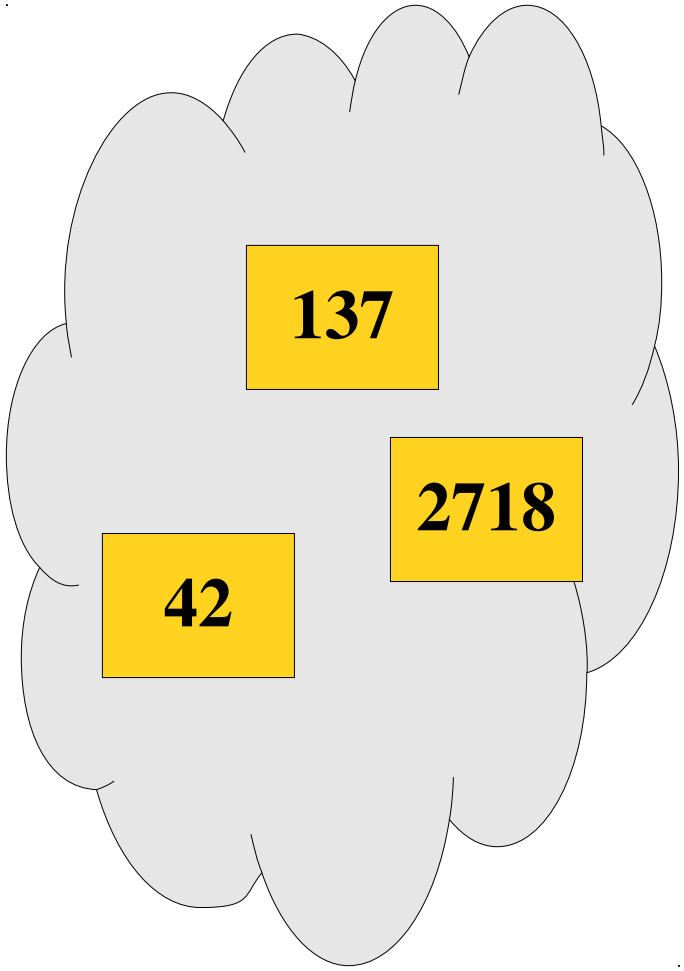
HashSet

- The **`HashSet`** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



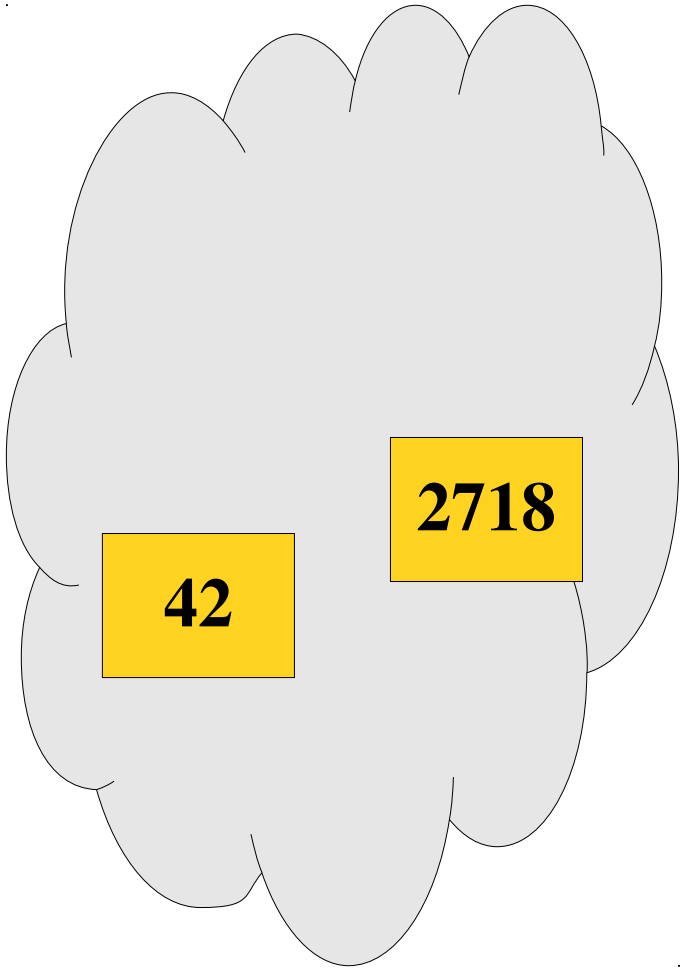
HashSet

- The **HashSet** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



HashSet

- The **HashSet** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.



Operations on `HashSets`

- You can add a value to a `HashSet` by writing

  hashSet `+=` ***value***`;`

- You can remove a value from a `HashSet` by writing

  hashSet `-=` ***value***`;`

- You can check if a value exists in a `HashSet` by writing

  hashSet`.contains(`***value***`)`

- Many more operations are available (union, intersection, difference, subset, etc.), so be sure to check the documentation.

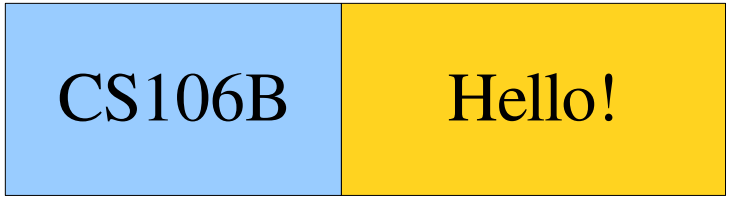
HashMap

HashMap

- The `HashMap` class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

HashMap

- The **HashMap** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.



HashMap

- The **HashMap** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

CS106B	Hello!
Dikdik	Cute!

HashMap

- The **HashMap** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

CS106B	Hello!
Dikdik	Cute!
This Slide	Self Referential

HashMap

- The **HashMap** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

CS106B	Hello!
Dikdik	**Very** Cute!
This Slide	Self Referential

Using the `Map`

- You can create a map by writing

  `HashMap<`***KeyType***`,` ***ValueType***`>` ***hashMap***`;`

- You can add or change a key/value pair by writing

  hashMap`[`***key***`] = `***value***`;`

  If the key doesn't already exist, it is added.

- You can read the value associated with a key by writing

  hashMap`[`***key***`]`

  If the key doesn't exist, it is added and associated with a default value. (More on that later.)

- You can check whether a key exists by calling

  hashMap`.containsKey(`***key***`)`

Using the `Map`

You can create a map by writing

```
HashMap<KeyType, ValueType> hashMap;
```

You can add or change a key/value pair by writing

```
hashMap[key] = value;
```

If the key doesn't already exist, it is added.

- You can read the value associated with a key by writing

```
hashMap[key]
```

If the key doesn't exist, it is added and associated with a default value. (More on that later.)

You can check whether a key exists by calling

```
hashMap.containsKey(key)
```

Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Map Autoinsertion

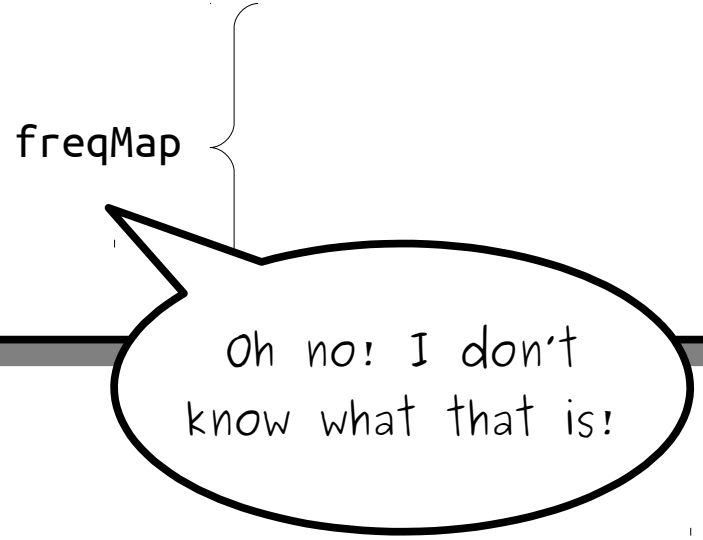
```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Map Autoinsertion

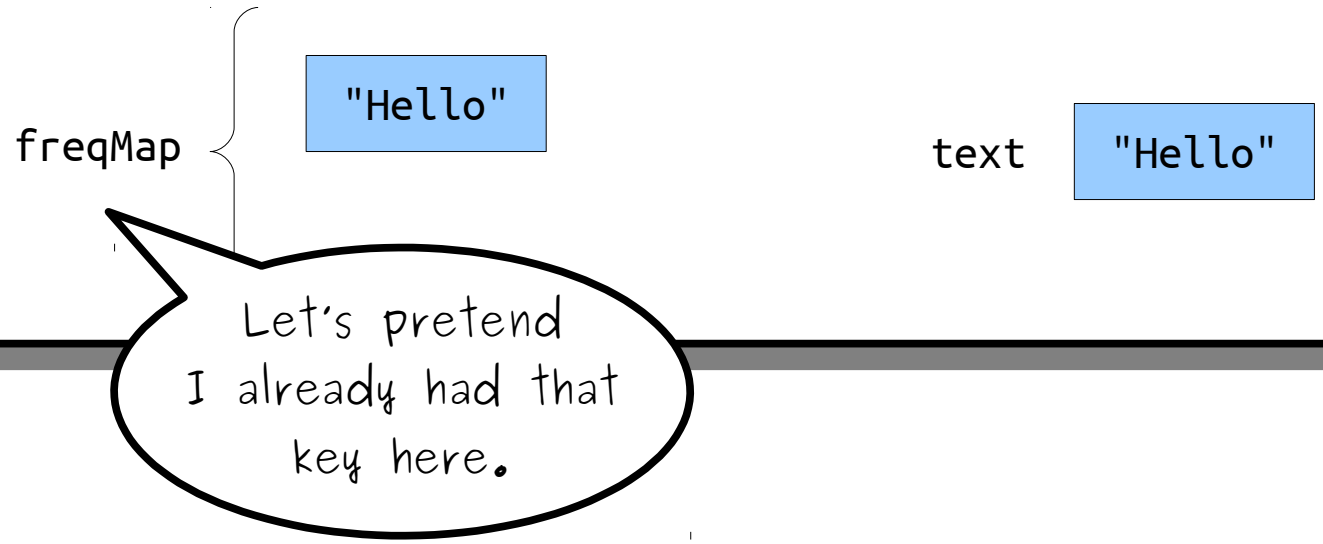
```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



text "Hello"

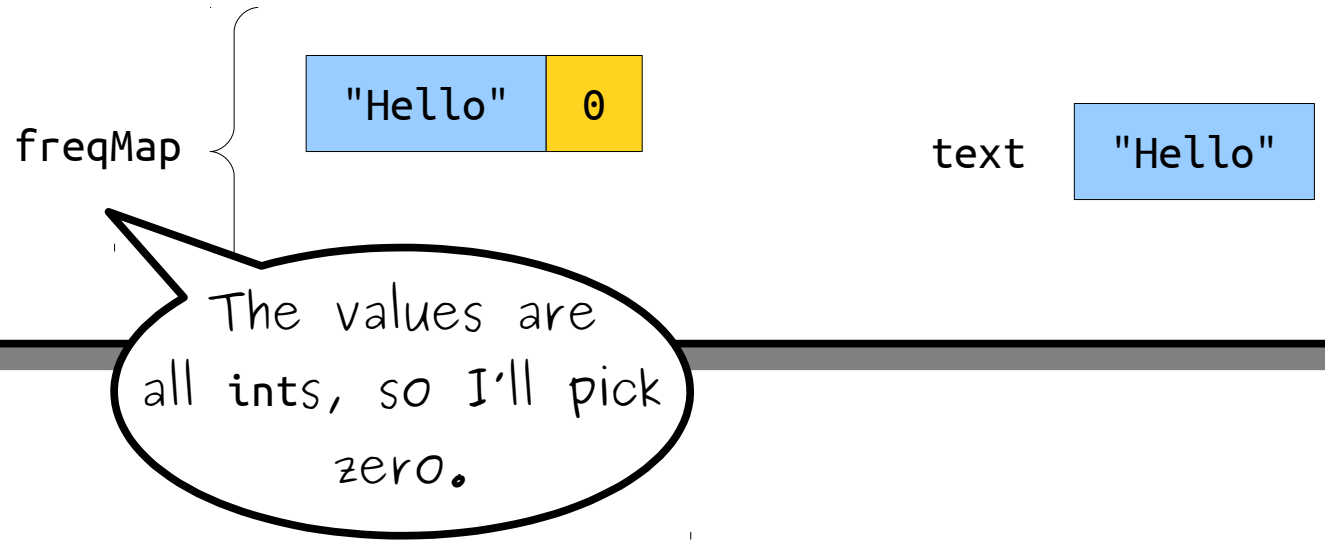
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



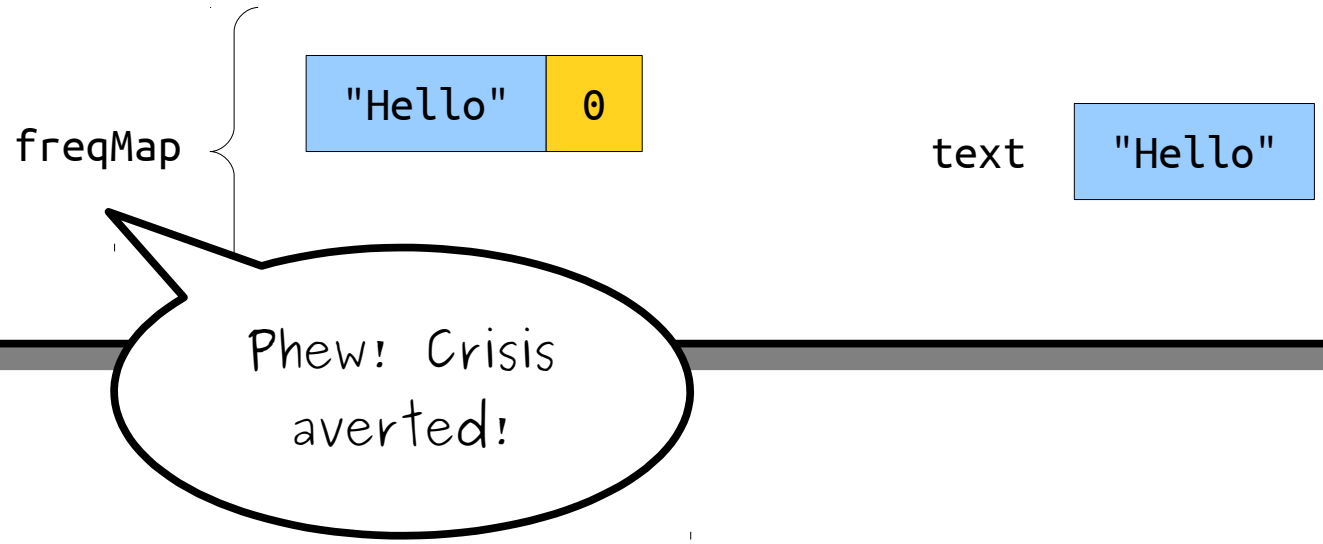
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



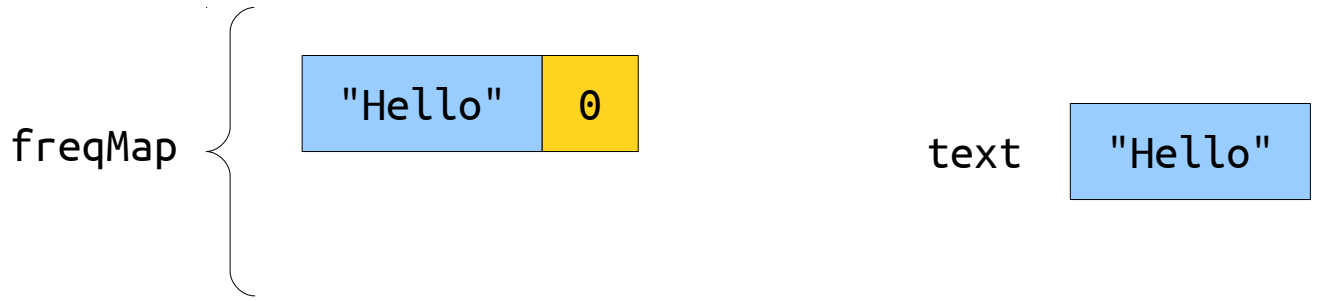
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



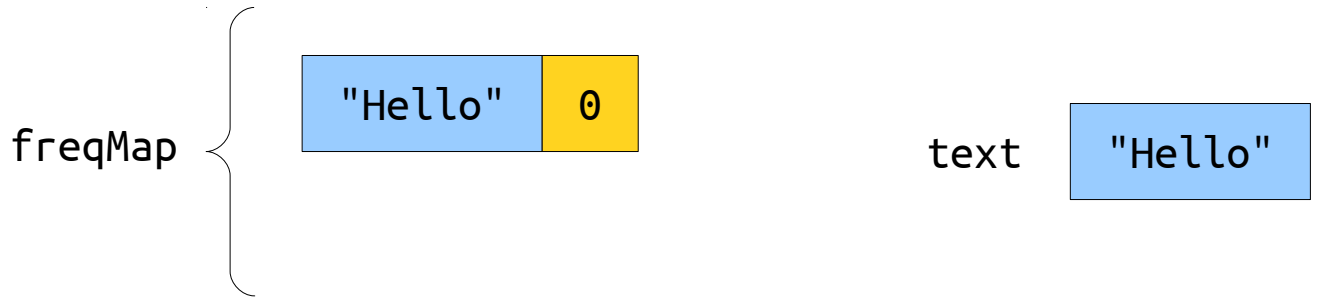
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



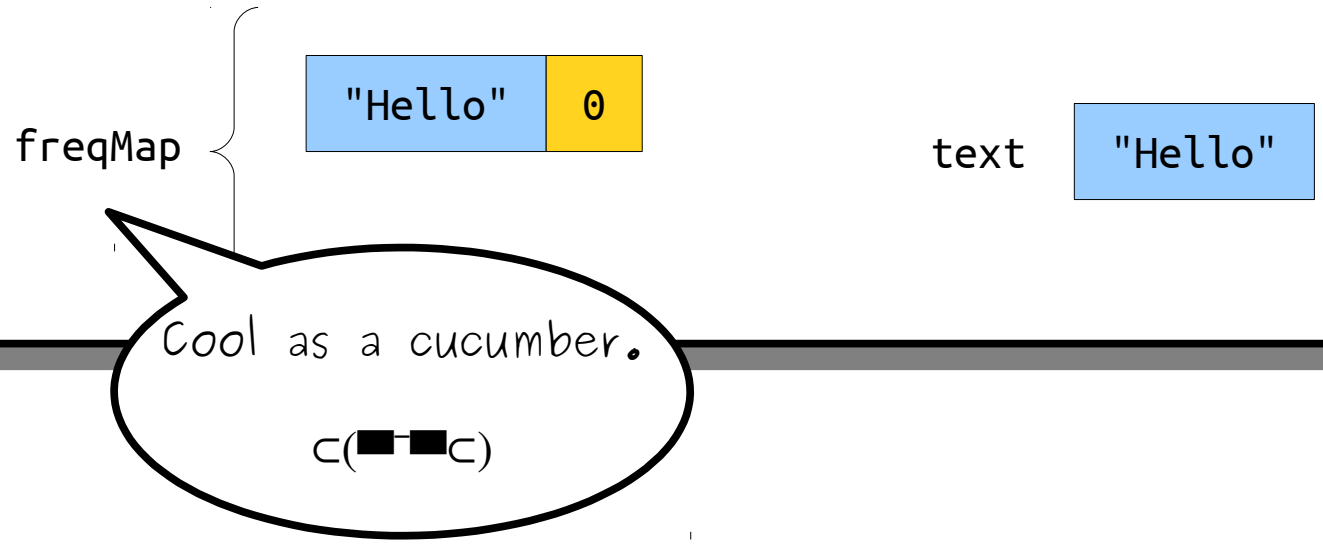
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



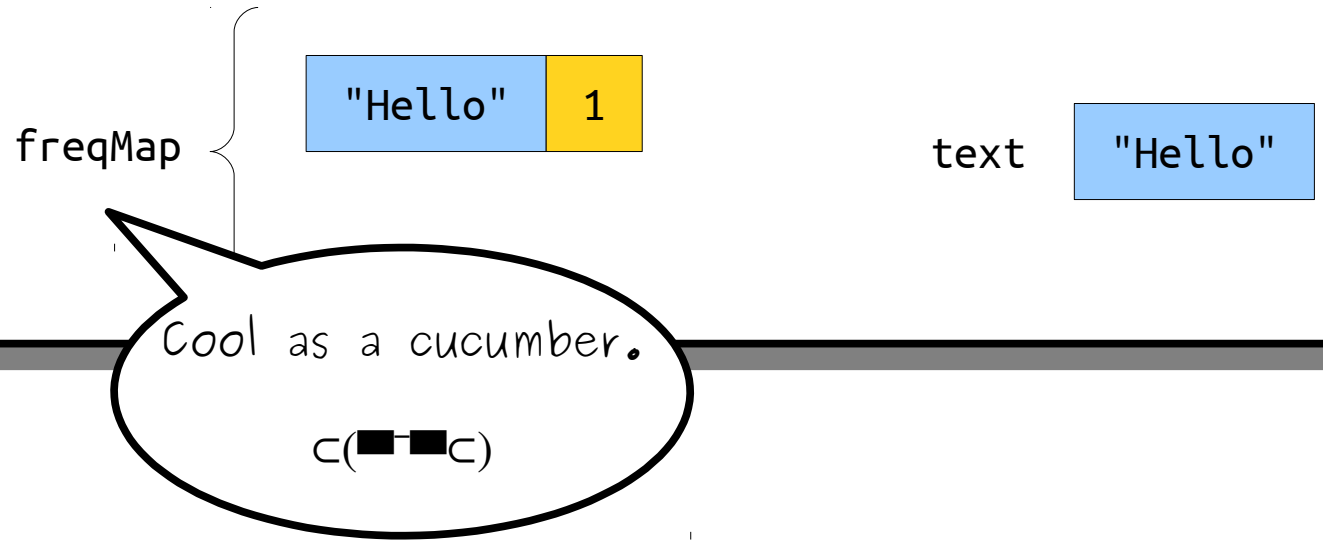
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



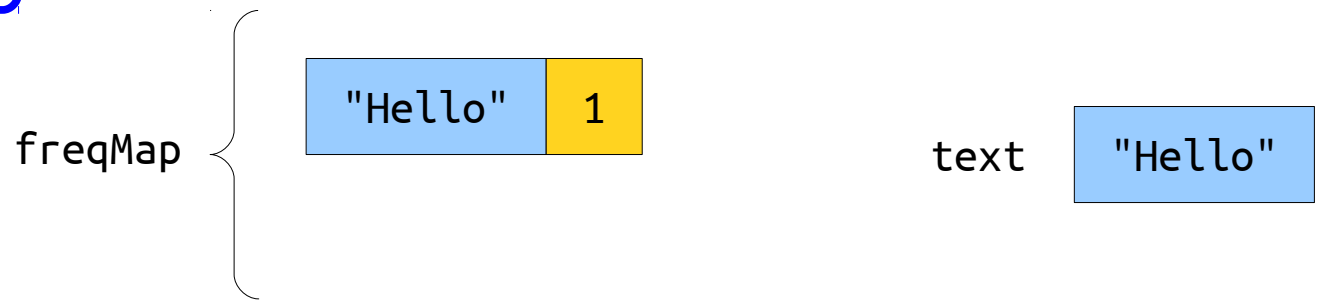
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



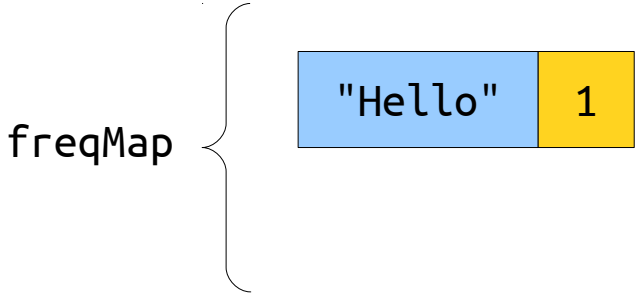
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



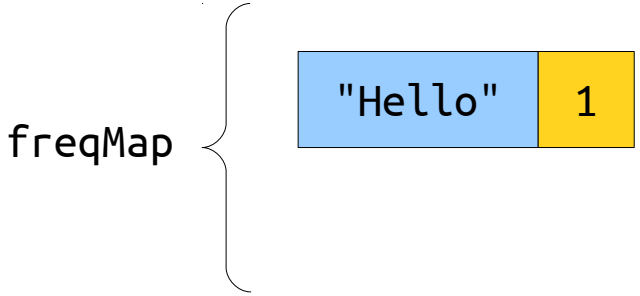
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



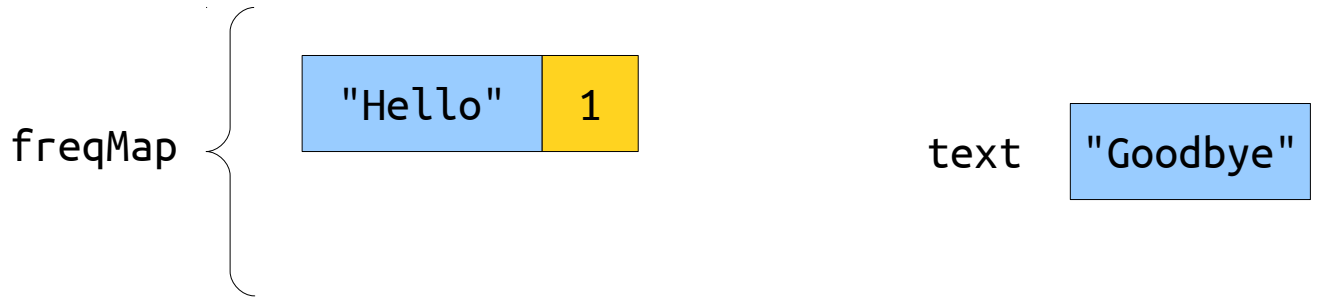
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



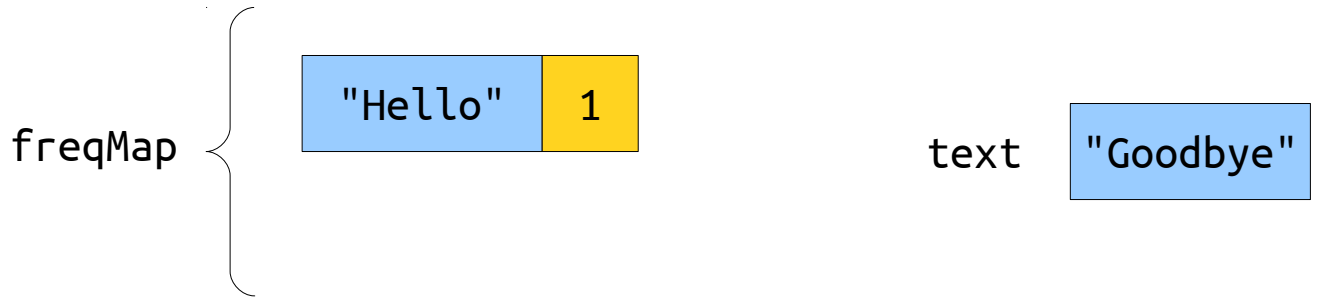
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



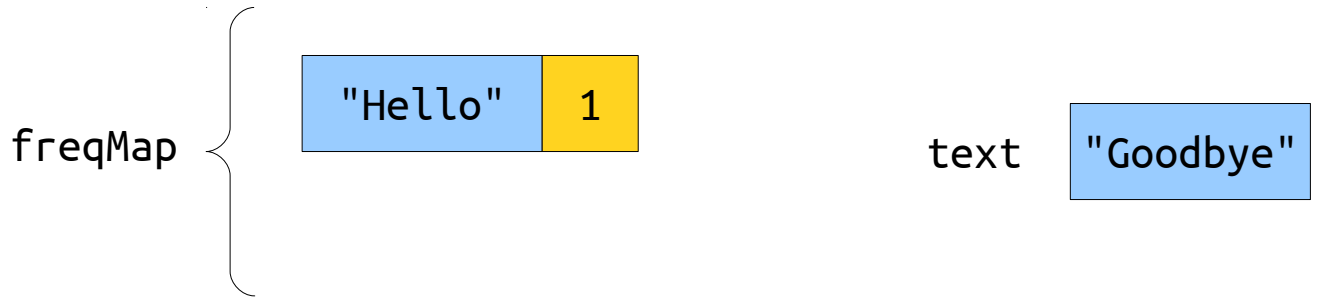
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



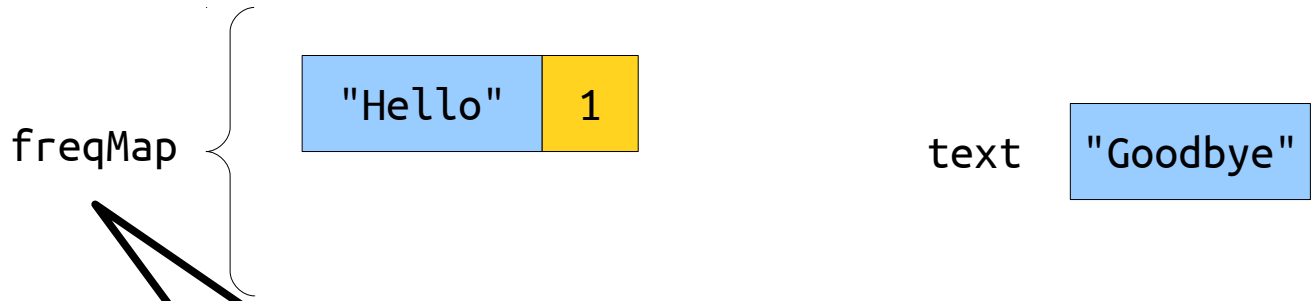
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



Map Autoinsertion

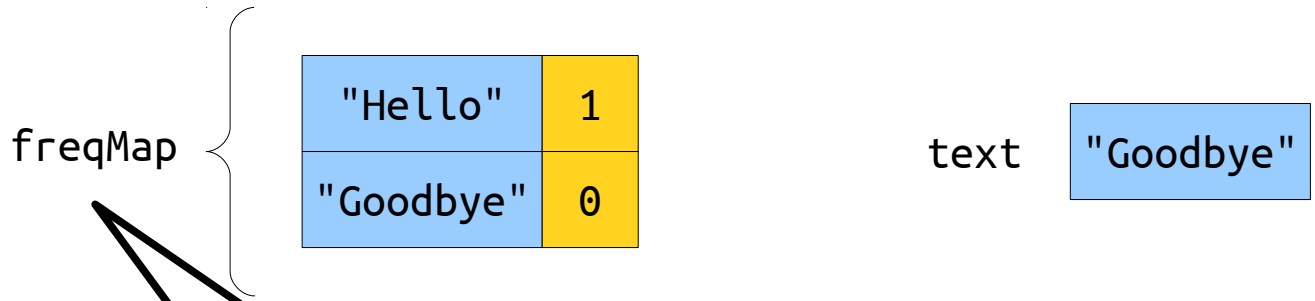
```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



Oh no, not again!

Map Autoinsertion

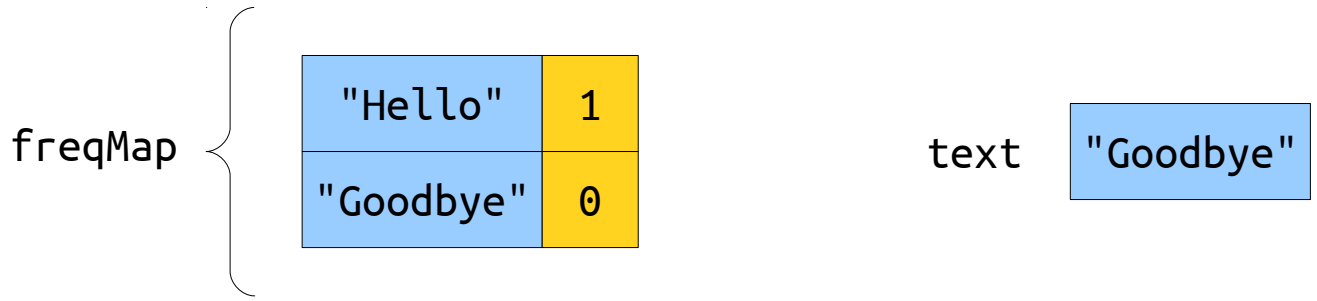
```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



I'll pretend I already had that key.

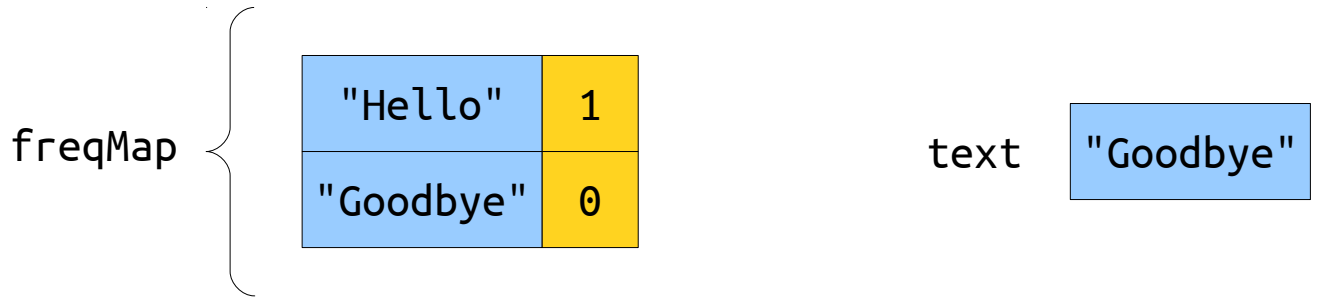
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

"Hello"	1
"Goodbye"	0

text "Goodbye"

Chillin' like a villain.

Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

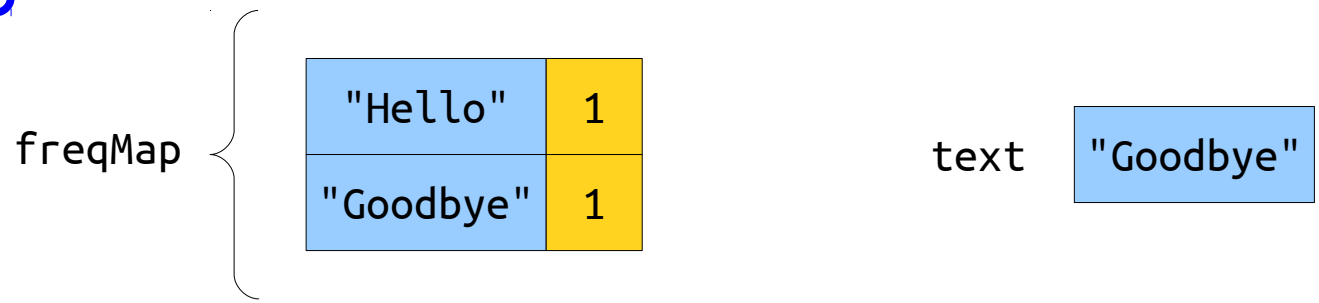
"Hello"	1
"Goodbye"	1

text "Goodbye"

Chillin' like a villain.

Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```



Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

"Hello"	1
"Goodbye"	1

Sorting by First Letters

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```

wordsByFirstLetter

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```

wordsByFirstLetter

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```

wordsByFirstLetter

word "first"

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```

wordsByFirstLetter

word "first"

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```

wordsByFirstLetter

word "first"

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```

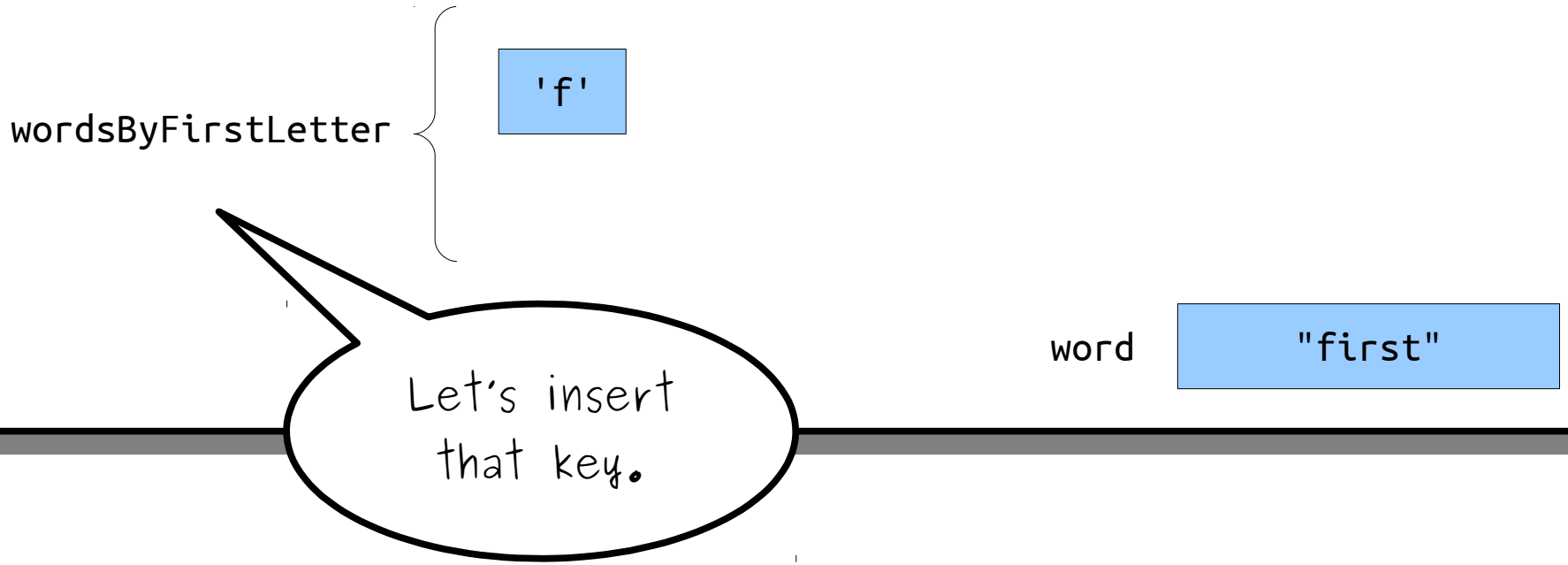
wordsByFirstLetter

word "first"

Oops, no f's here.

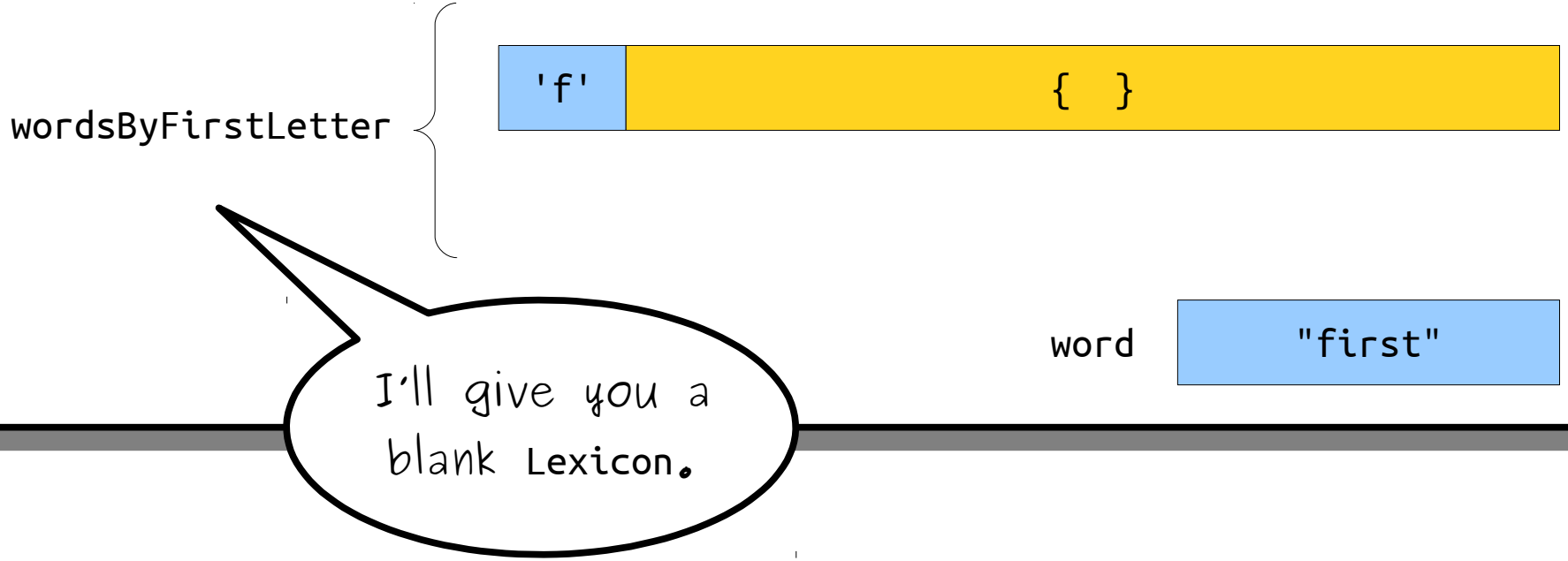
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```



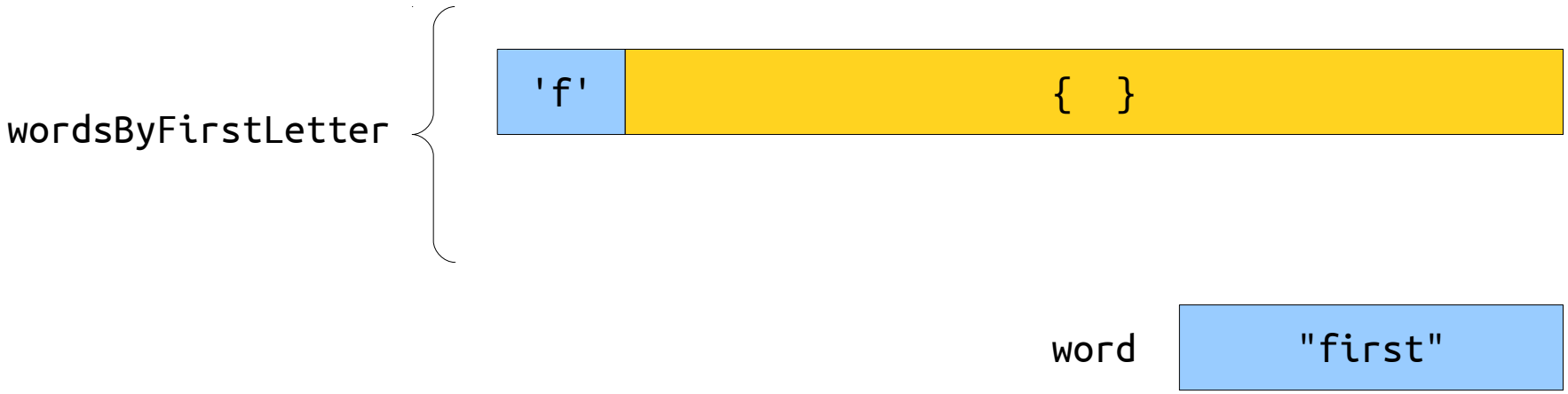
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



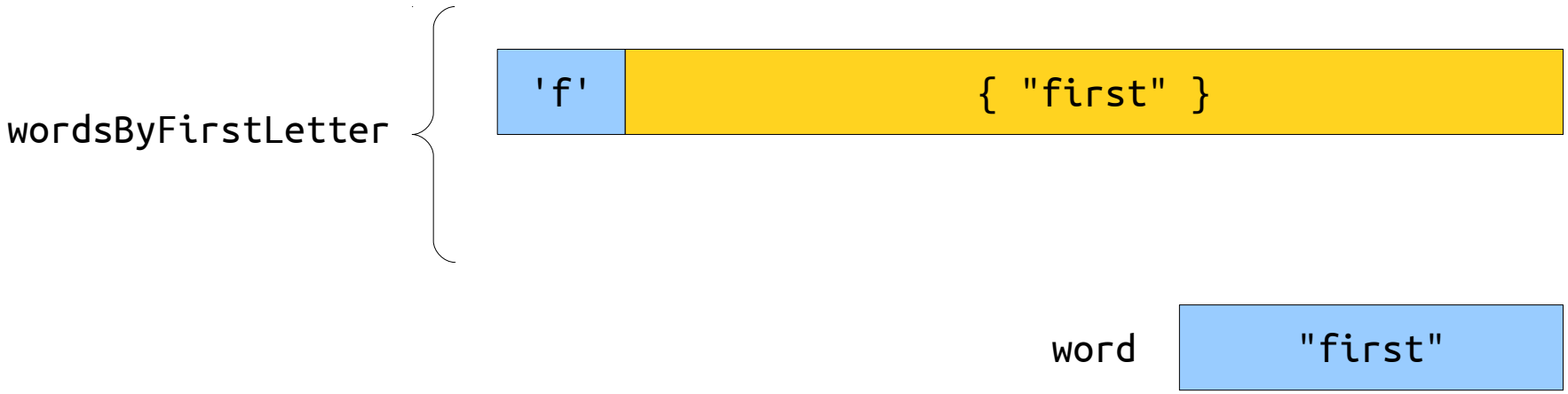
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```



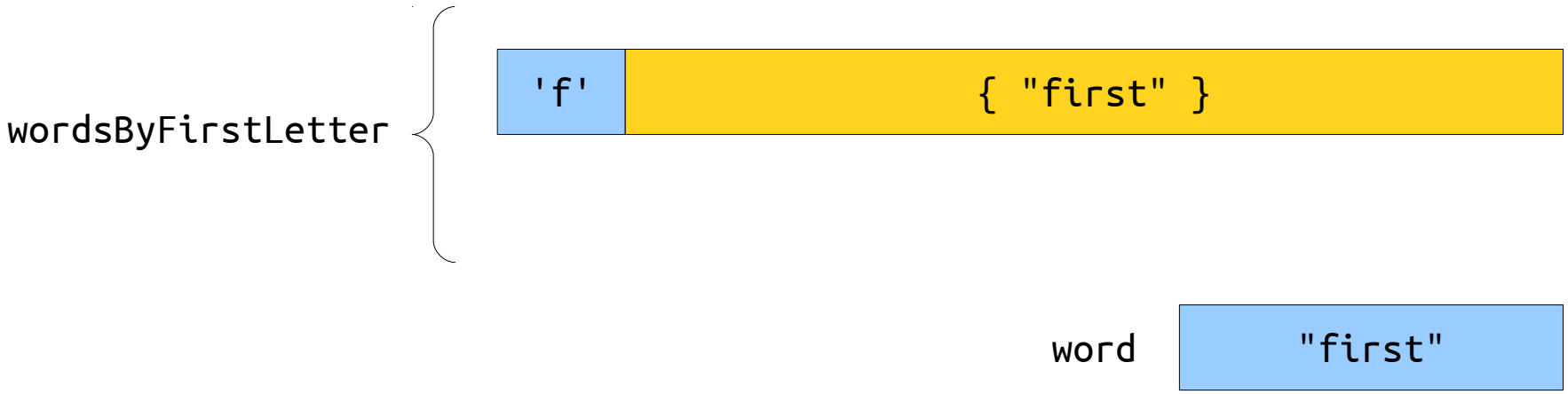
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```



Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



Map Autoinsertion

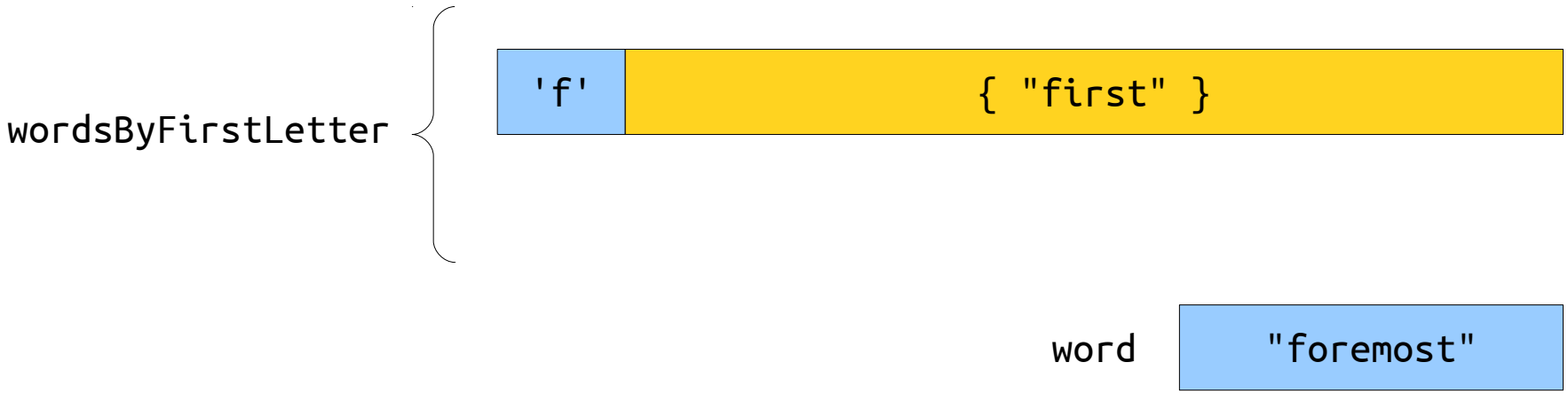
```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```

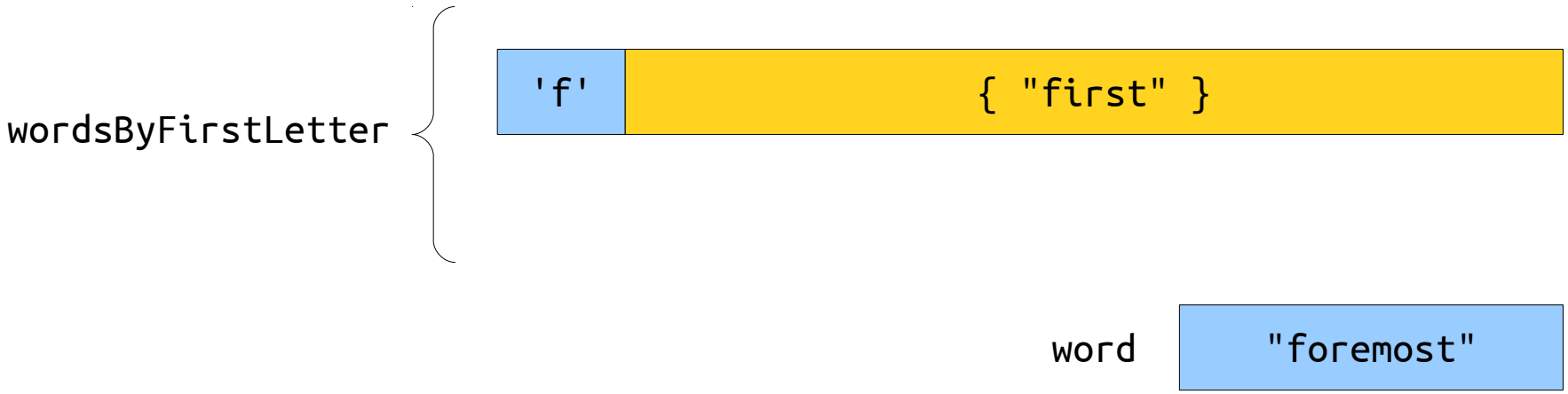
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



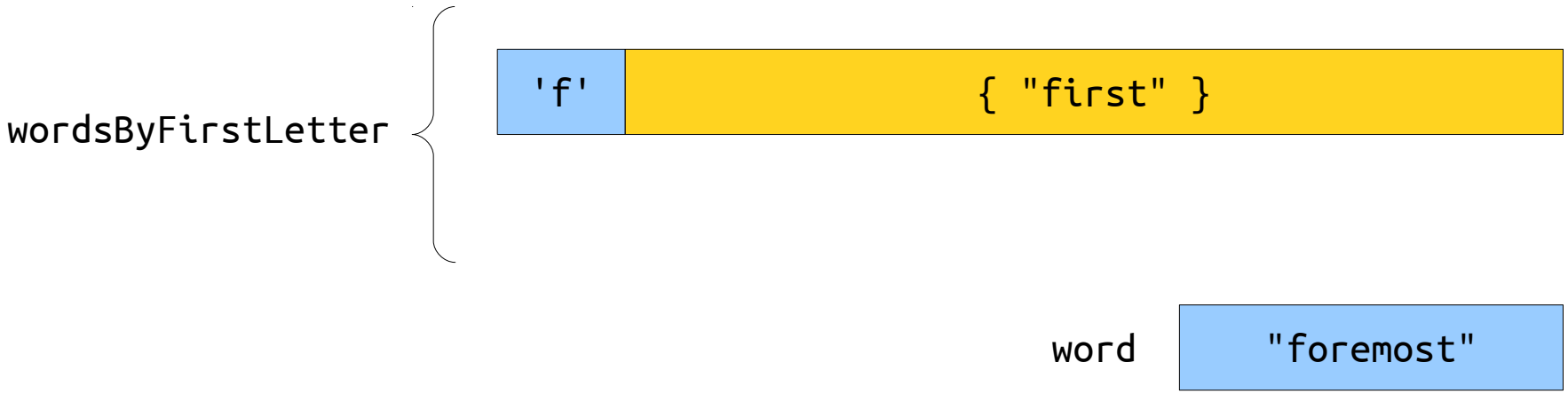
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



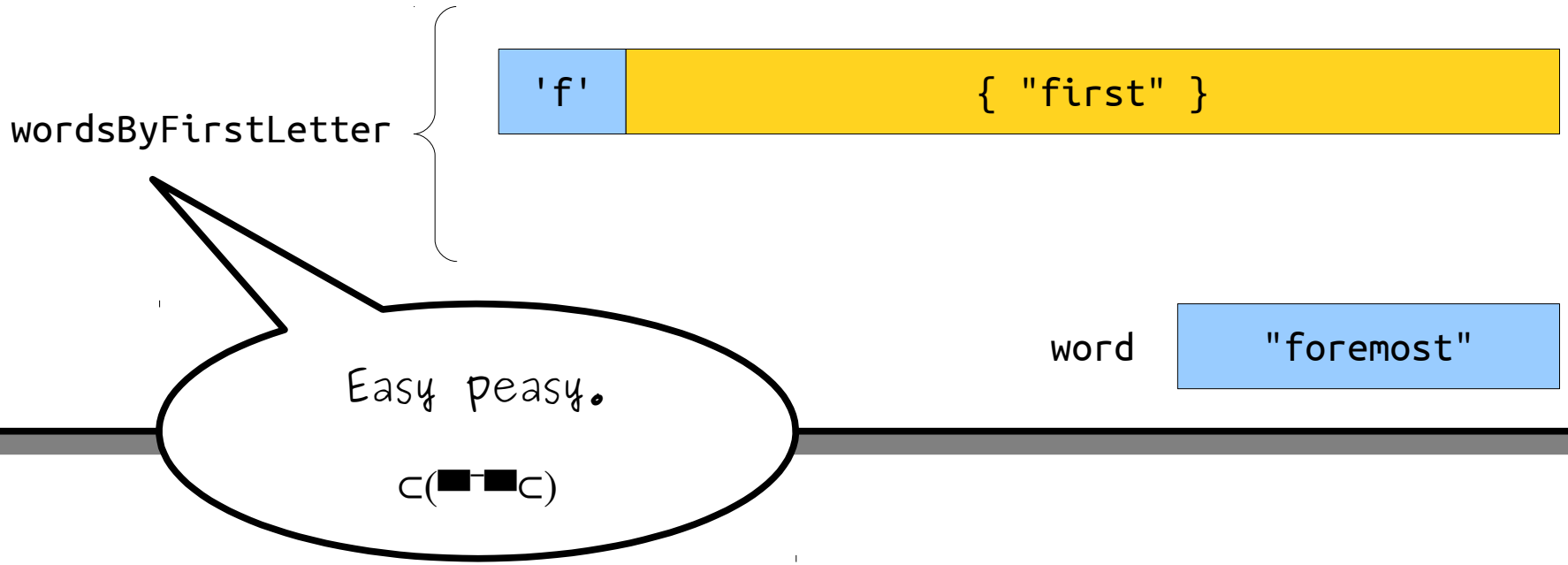
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



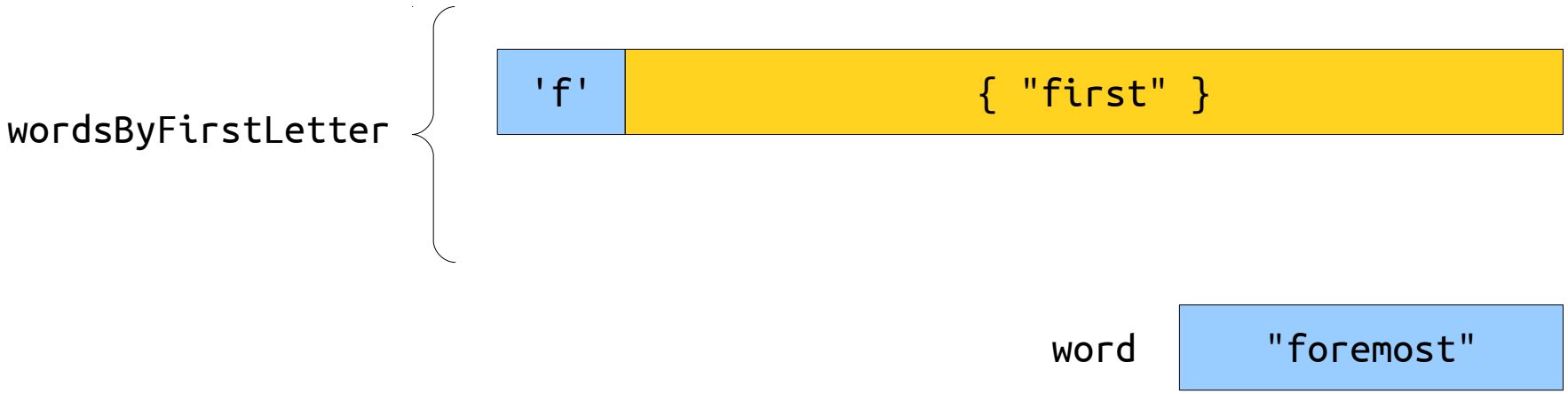
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



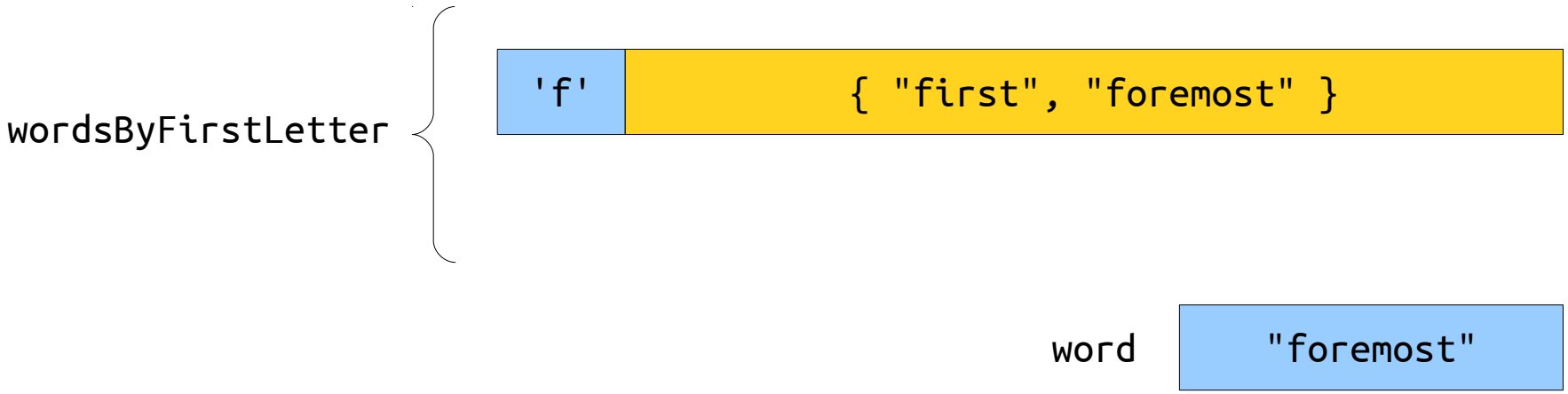
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```



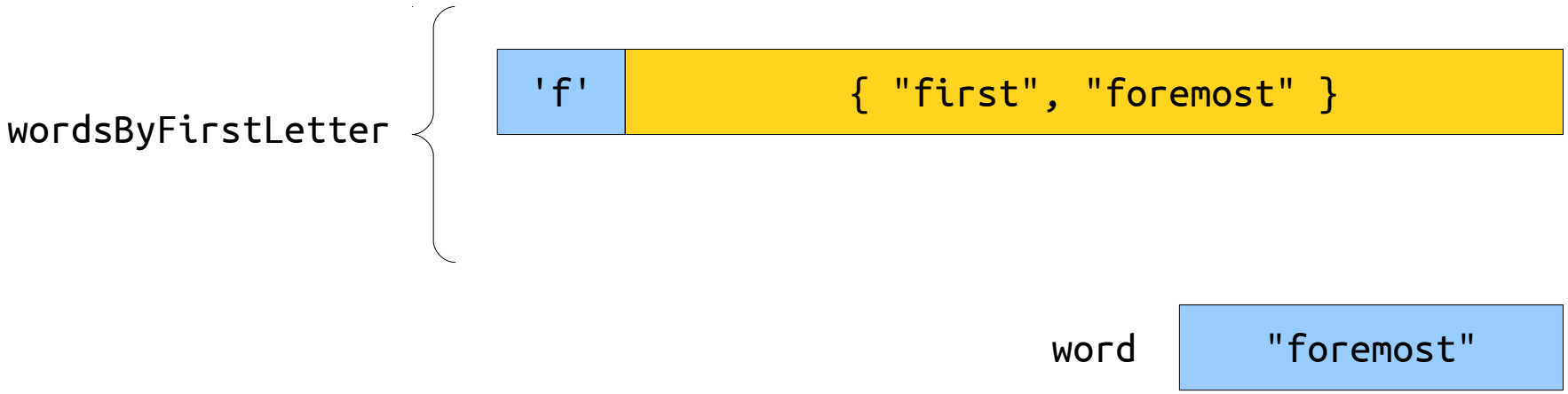
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```



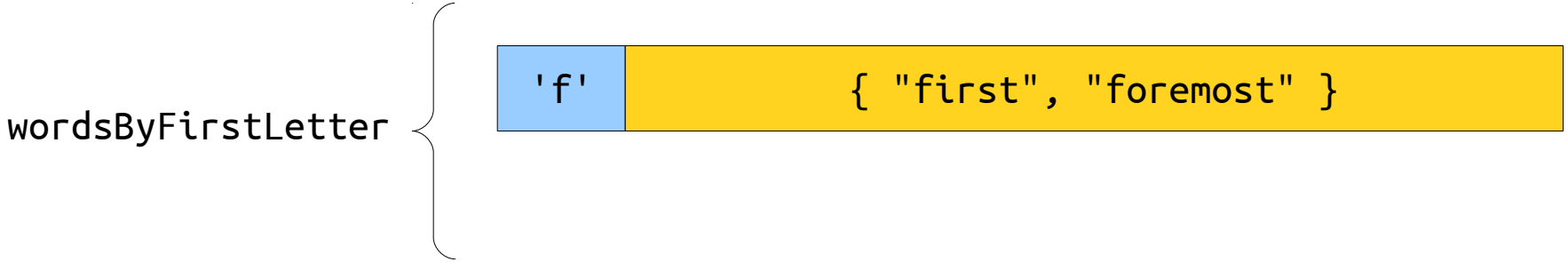
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



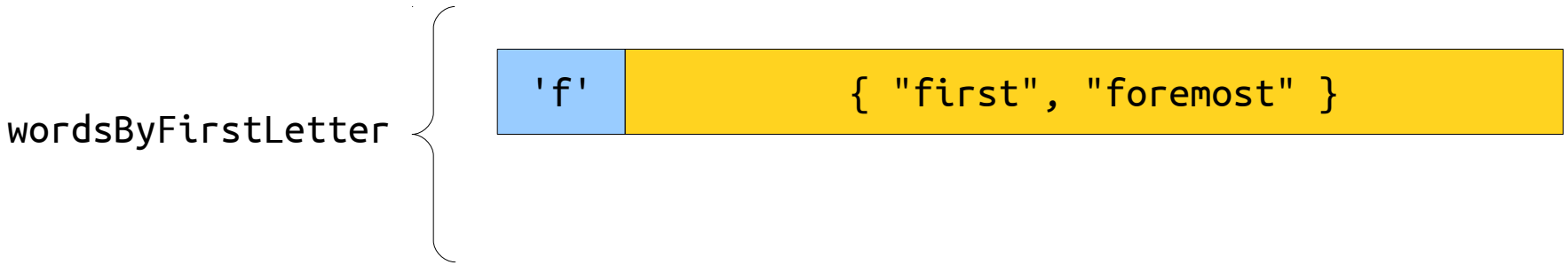
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



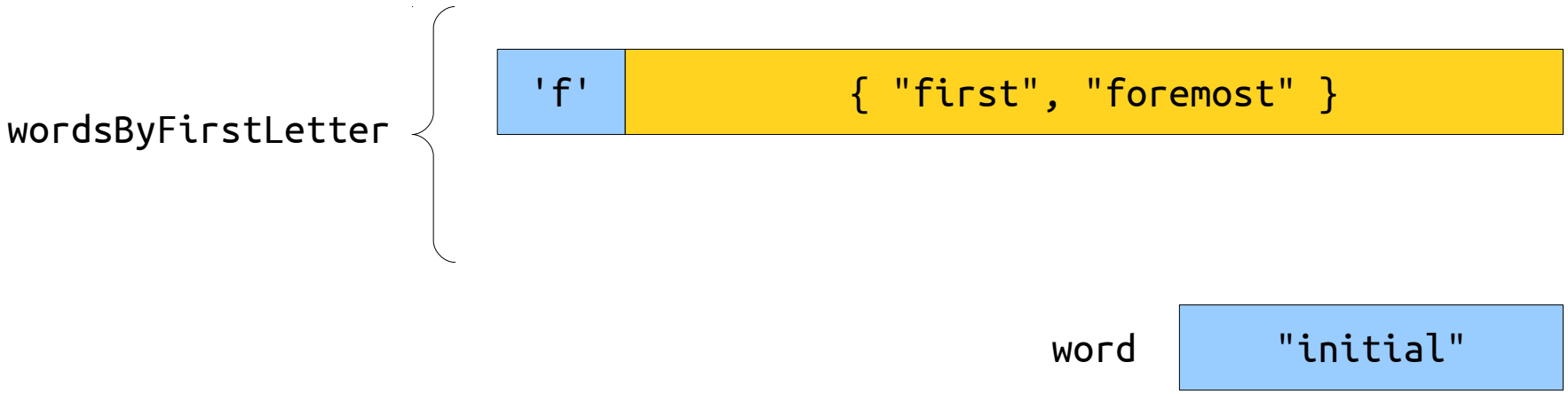
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



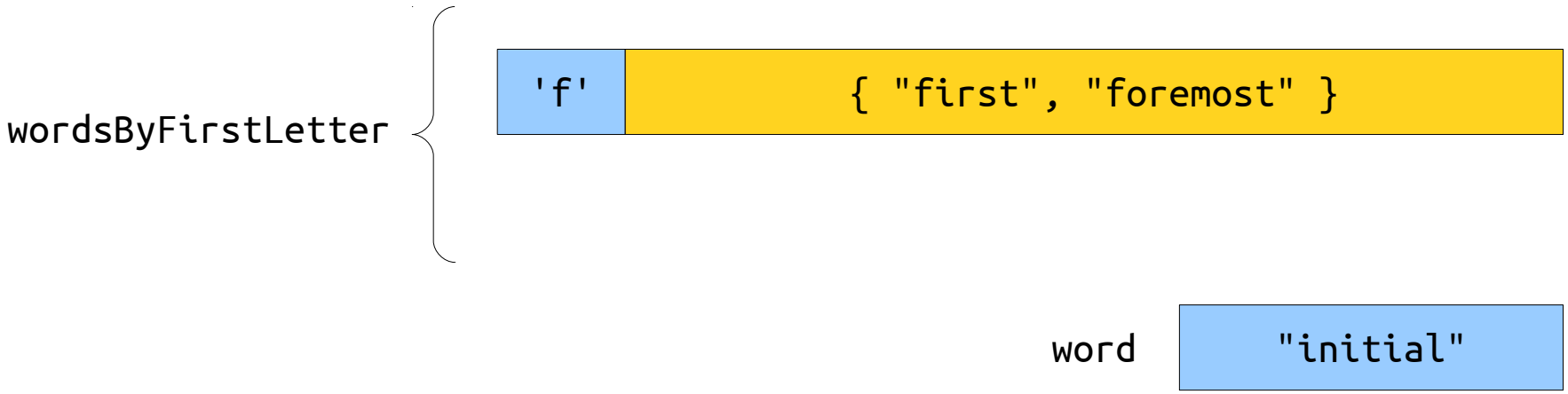
Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```



Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```

wordsByFirstLetter

'f'	{ "first", "foremost" }
'i'	{ }

word "initial"

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```

wordsByFirstLetter

'f'	{ "first", "foremost" }
'i'	{ }

word "initial"

Map Autoinsertion

```
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
   wordsByFirstLetter[word[0]].add(word);
}
```

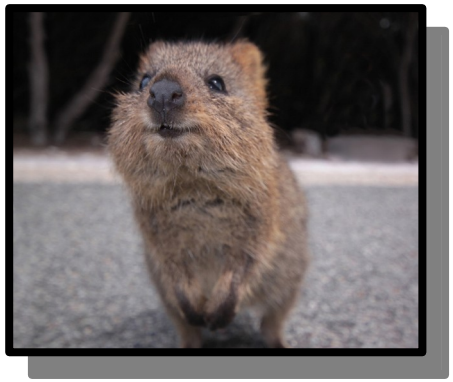
wordsByFirstLetter

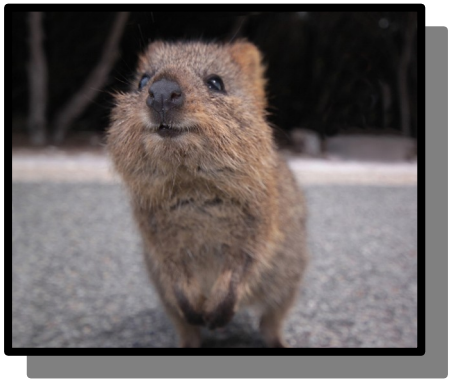
'f'	{ "first", "foremost" }
'i'	{ "initial" }

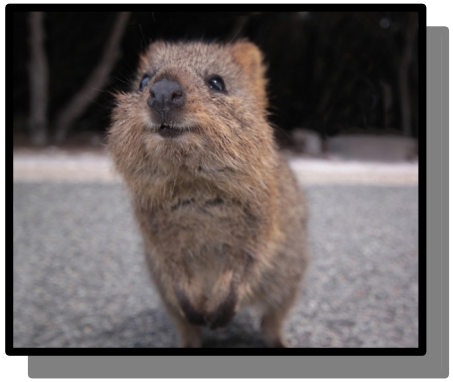
word "initial"

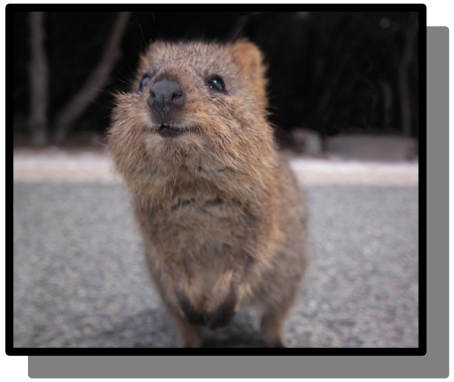
Quokka

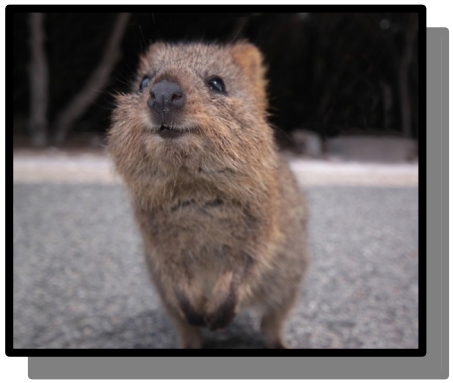
Quokka Quincunx



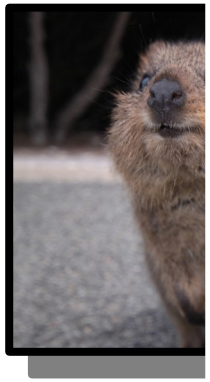


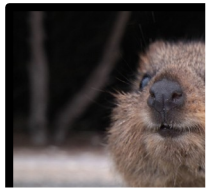






Quarter Quokka Quincunx





Anagrams

Anagrams

- Two words are ***anagrams*** of one another if the letters in one can be rearranged into the other.
- Some examples:
  - “Praising” and “aspiring.”
  - “Arrogant” and “tarragon.”
- ***Question for you:*** does this concept exist in other languages? If so, please send me examples!

Anagrams

- ***Nifty fact:*** two words are anagrams if you get the same string when you write the letters in those words in sorted order.
- For example, “praising” and “aspiring” are anagrams because, in both cases, you get the string “aiignprs” if you sort the letters.

Anagram Clusters

- Let's group all words in English into "clusters" of words that are all anagrams of one another.
- We'll use a `HashMap<string, Lexicon>`.
  - Each key is a string of letters in sorted order.
  - Each value is the collection of English words that have those letters in that order.

Your Action Items

- ***Read Chapter 5.***
  - It's all about container types, and it'll fill in any remaining gaps from this week.
- ***Start Assignment 2.***
  - Make slow and steady progress here, if you can. Aim to complete Rising Tides and to have started You Got Hufflepuff!
- ***Enjoy MLK Weekend***
  - Read "Letter from Birmingham City Jail." Like, seriously.
  - Watch the actual "I Have a Dream" speech. It's truly amazing.
  - Watch "The Two Americas," a speech MLK delivered here on Stanford campus.

Next Time

- ***Thinking Recursively***
  - How can you best solve problems using recursion?
  - What techniques are necessary to do so?
  - And what problems yield easily to a recursive solution?

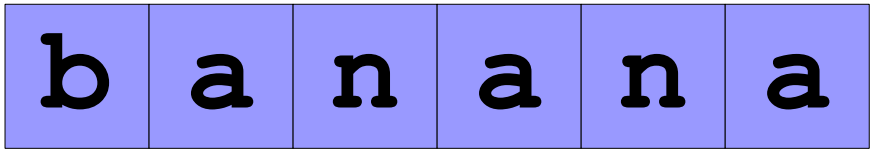
Appendix: How to Sort a String

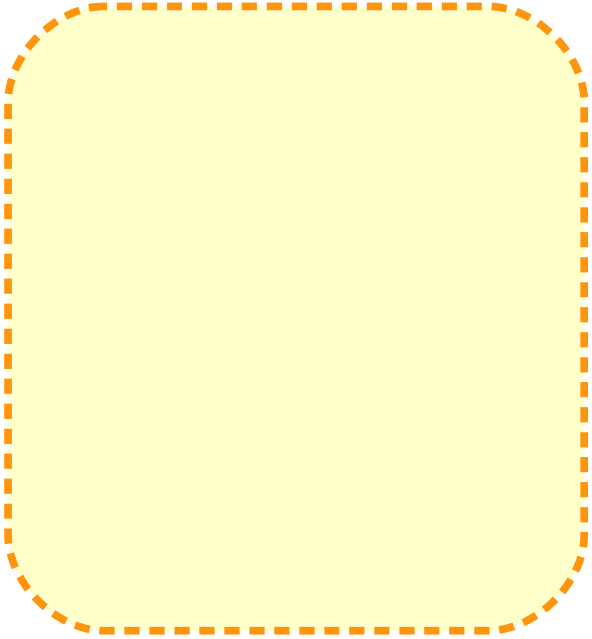
Counting Sort

Counting Sort

b	a	n	a	n	a

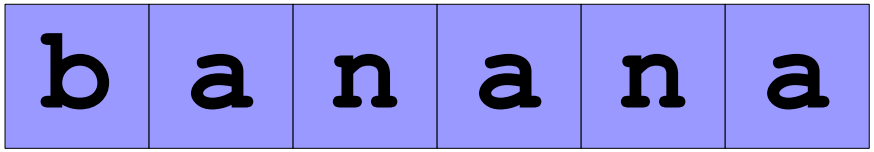
Counting Sort

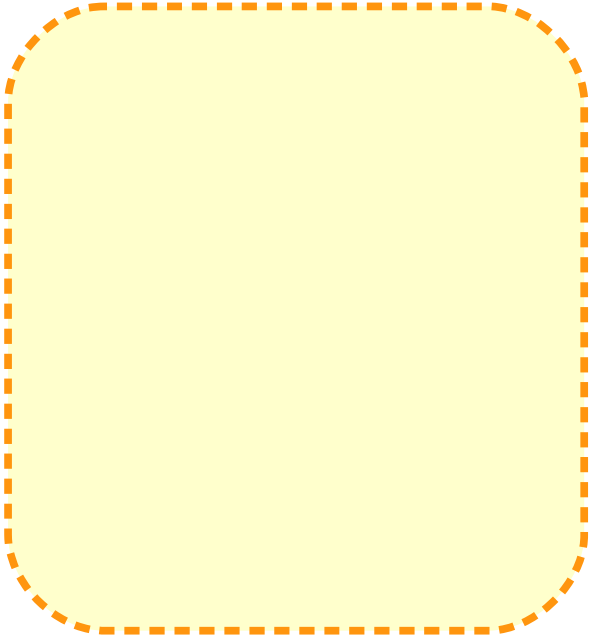




letterFreq

Counting Sort

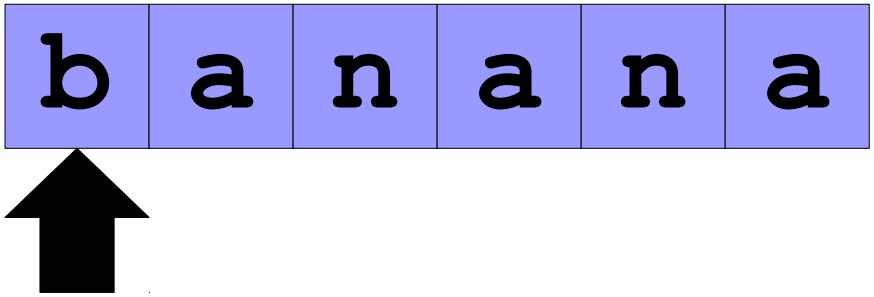


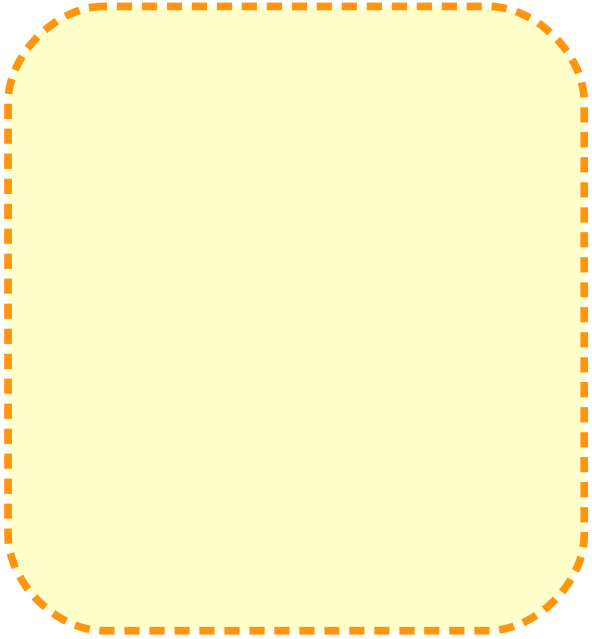


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

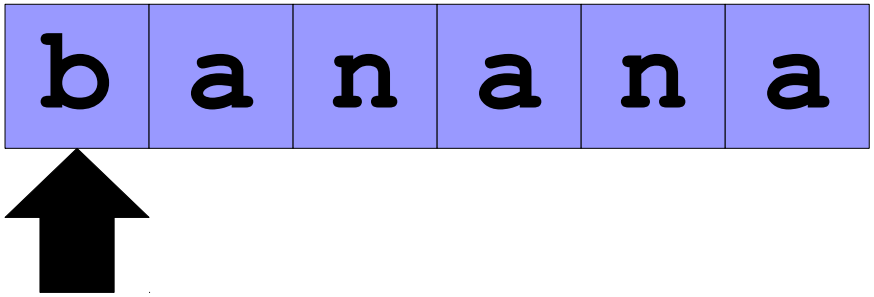


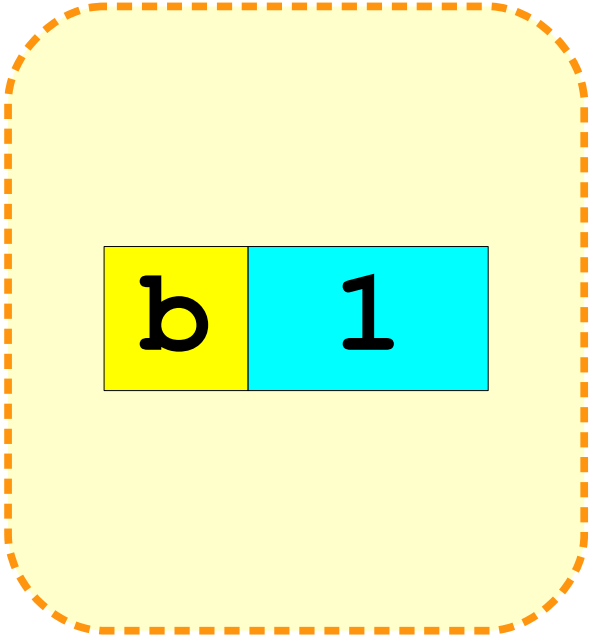


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

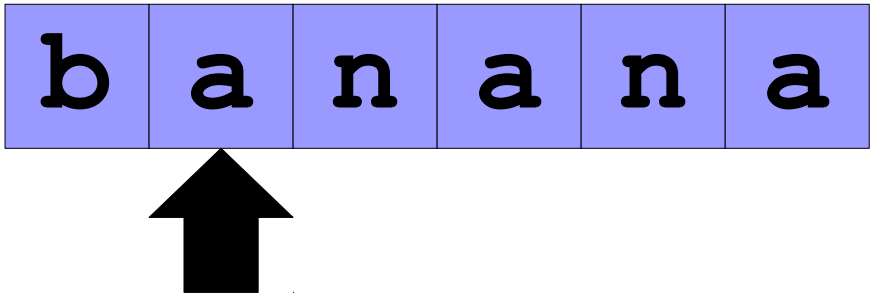


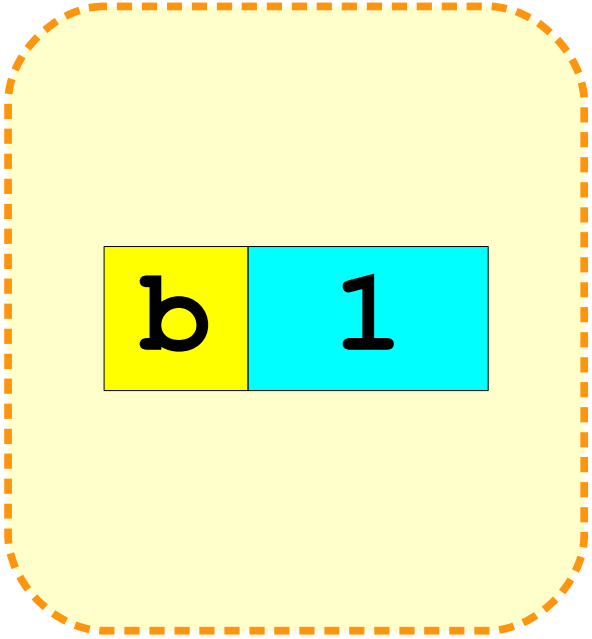


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

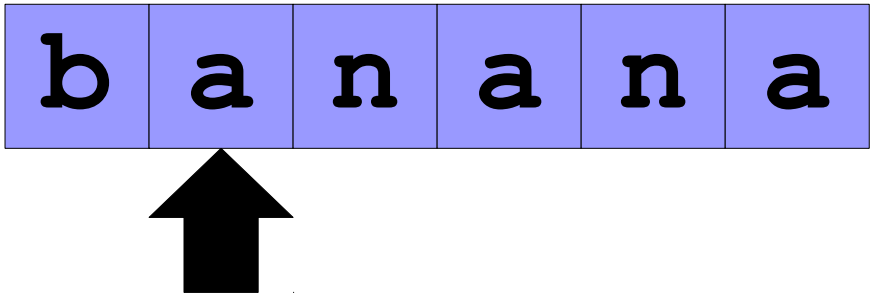


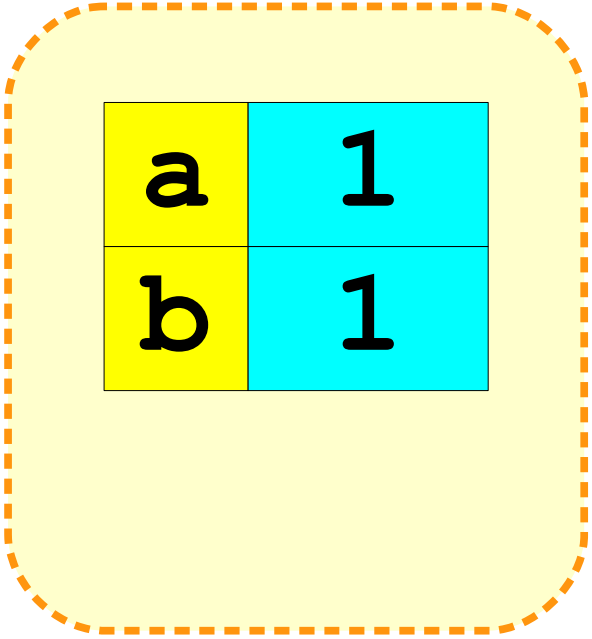


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

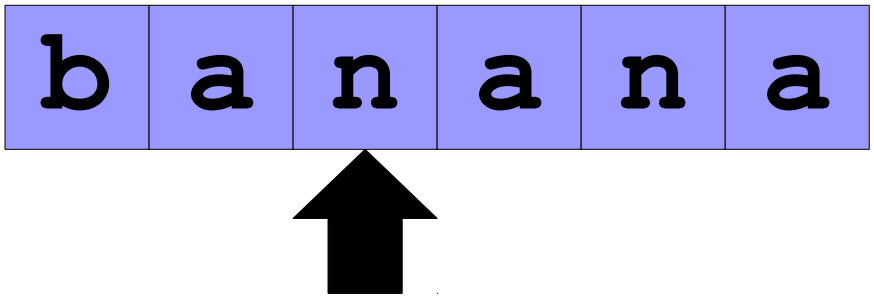


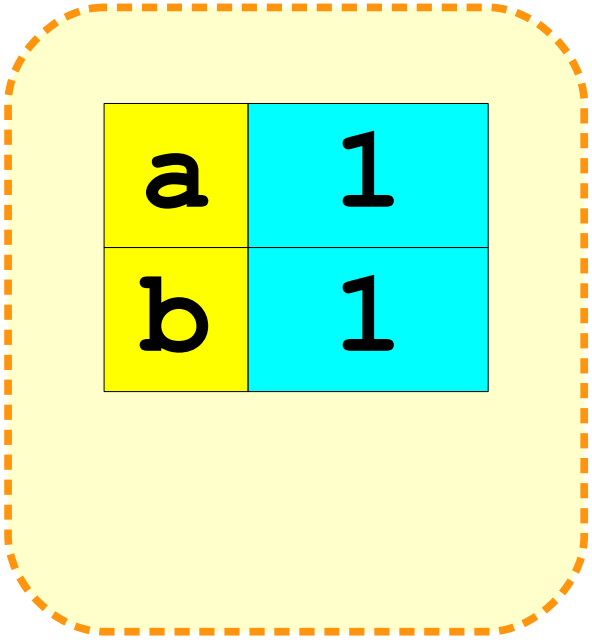


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

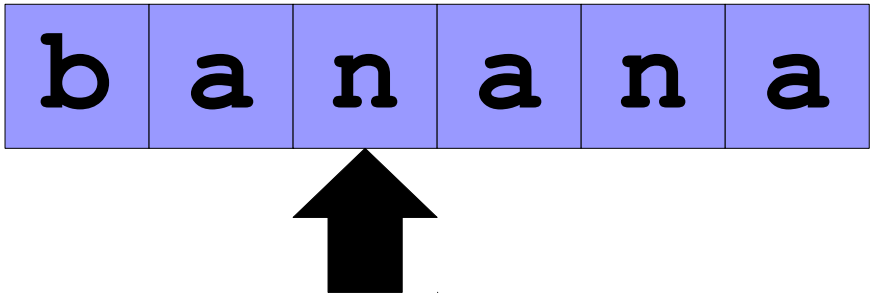


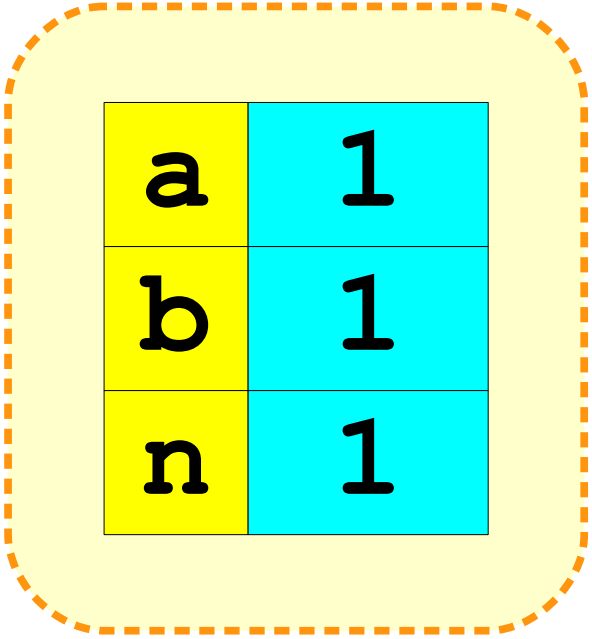


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

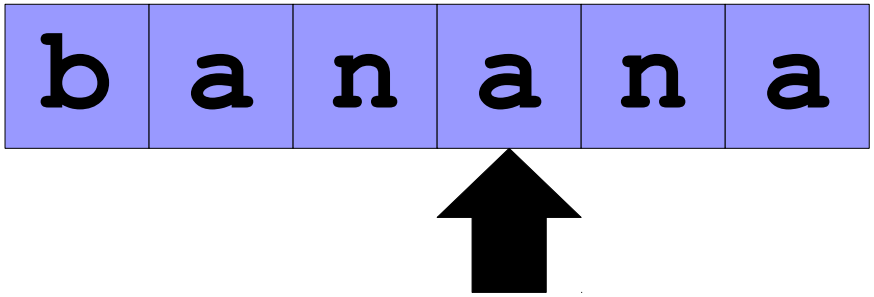


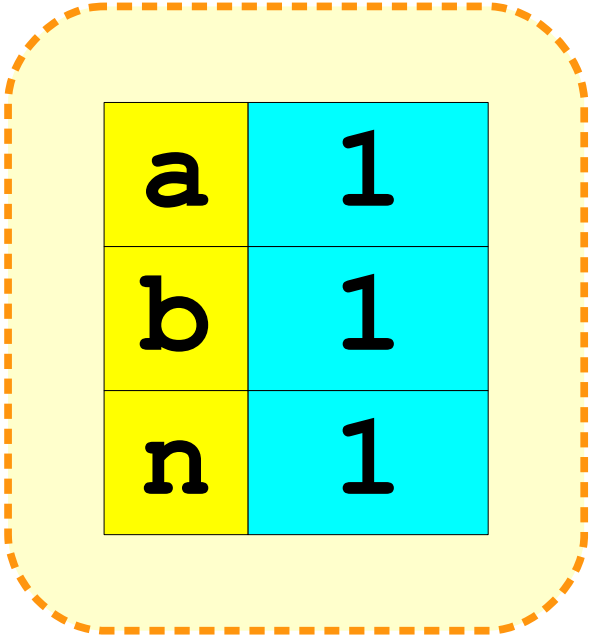


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

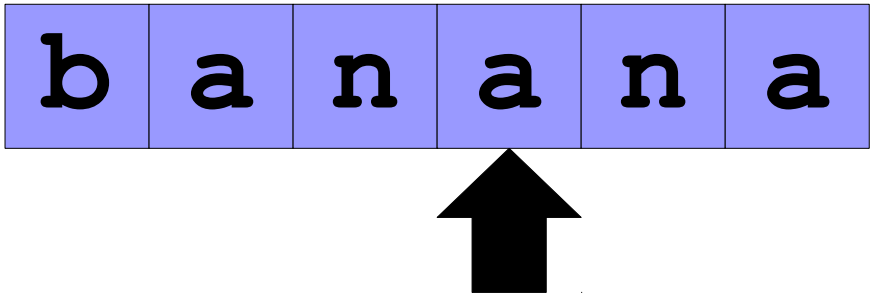


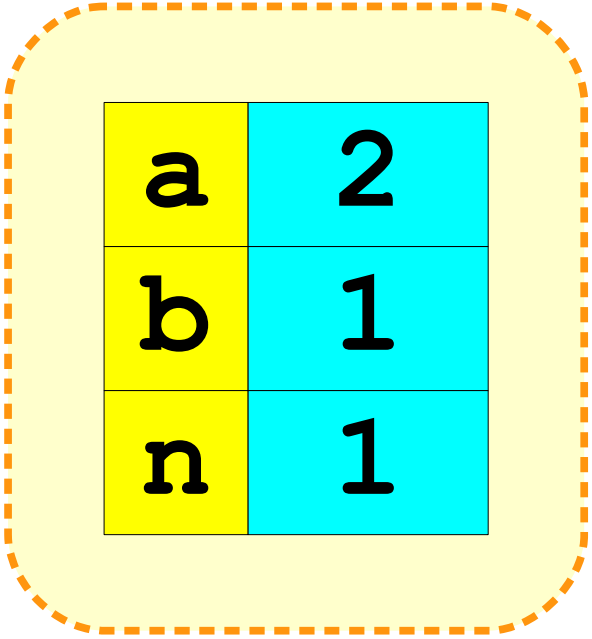


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

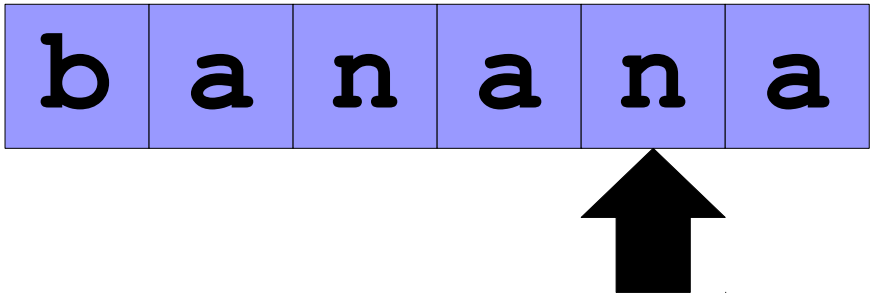


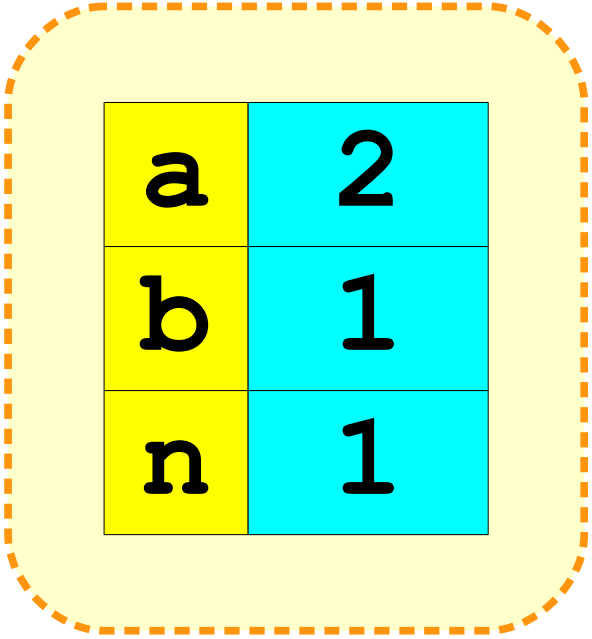


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

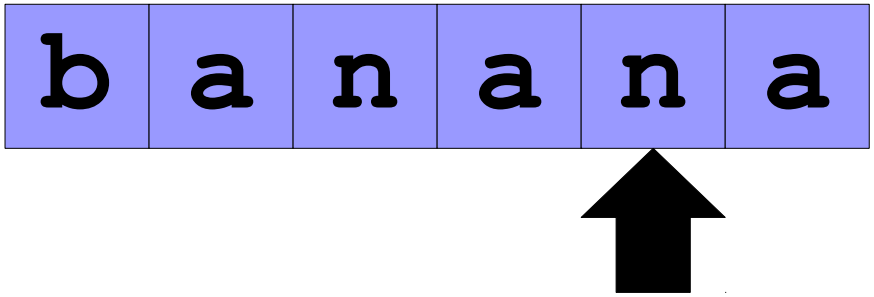


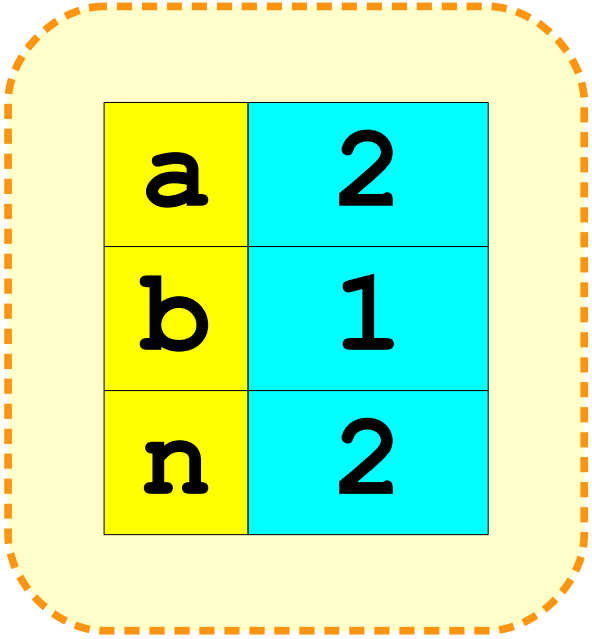


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

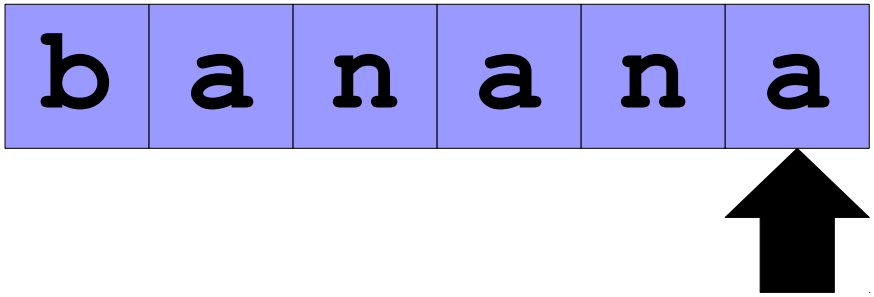


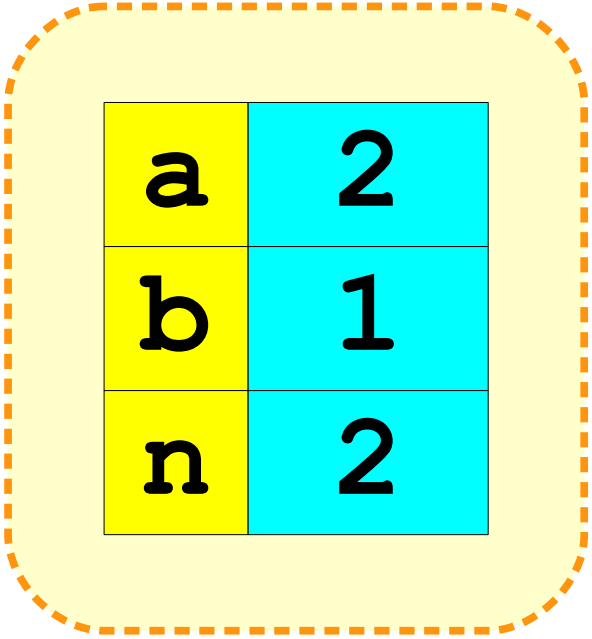


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

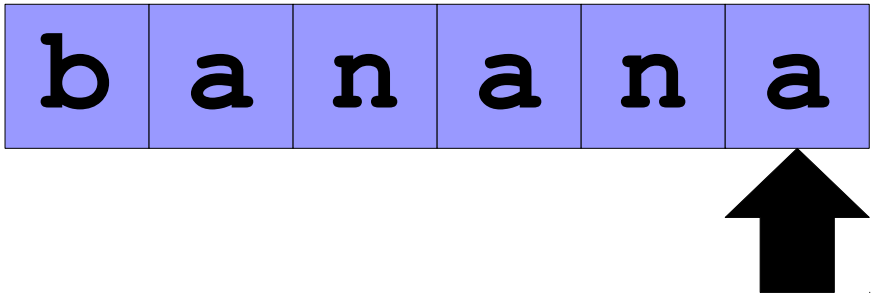


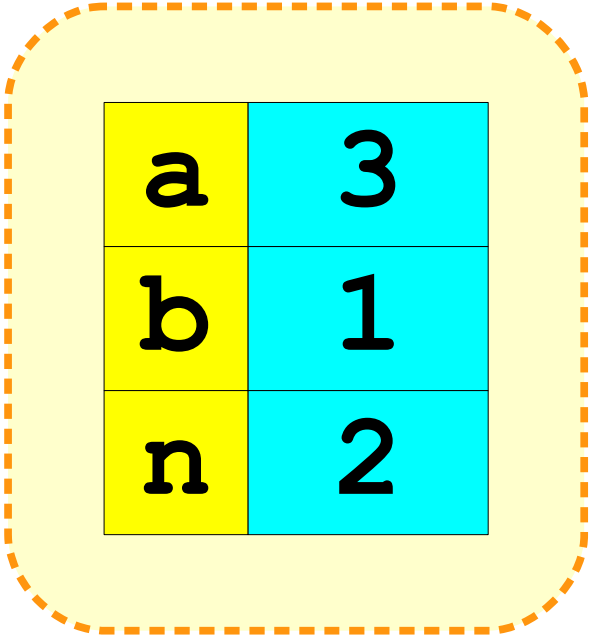


letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort





letterFreq

```
for (char ch: input) {
    letterFreq[ch]++;
}
```

Counting Sort

b	a	n	a	n	a

a	3
b	1
n	2

letterFreq

Counting Sort

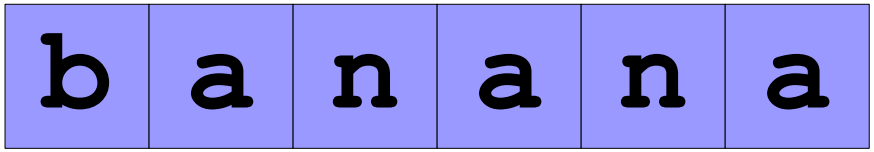
b	a	n	a	n	a

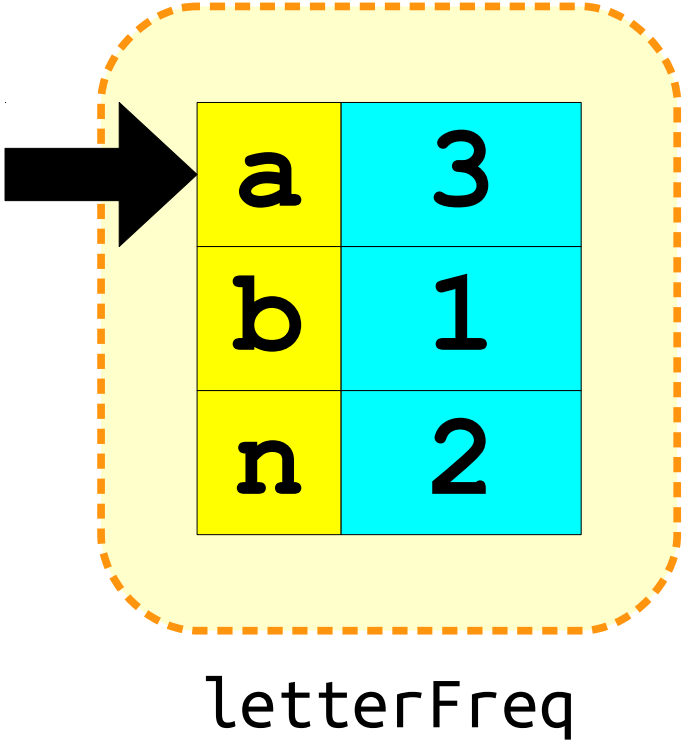
a	3
b	1
n	2

letterFreq

```
for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```

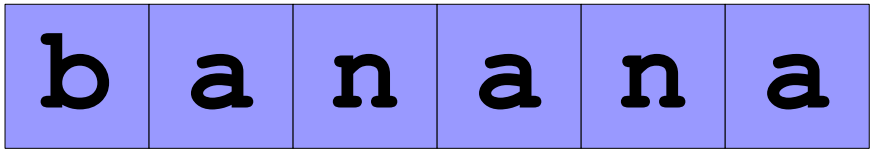
Counting Sort

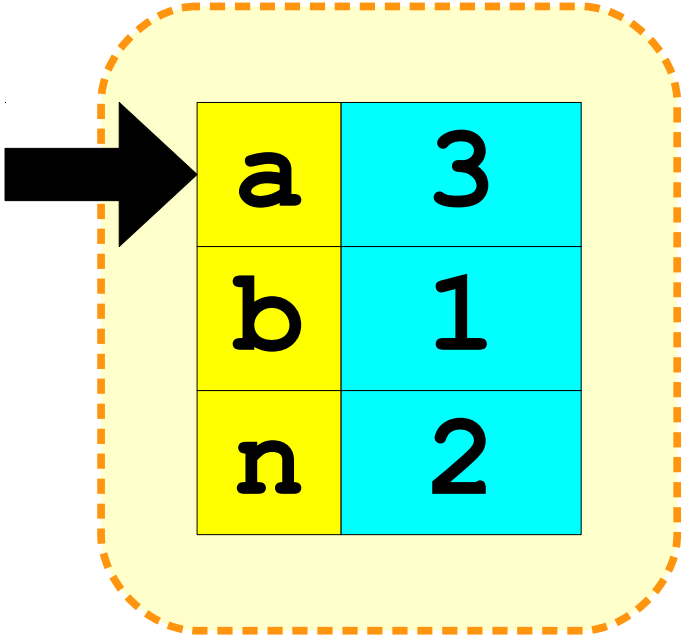





```
for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```

Counting Sort

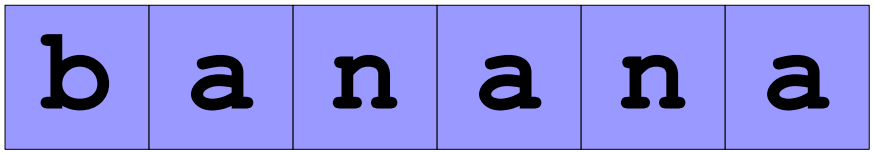


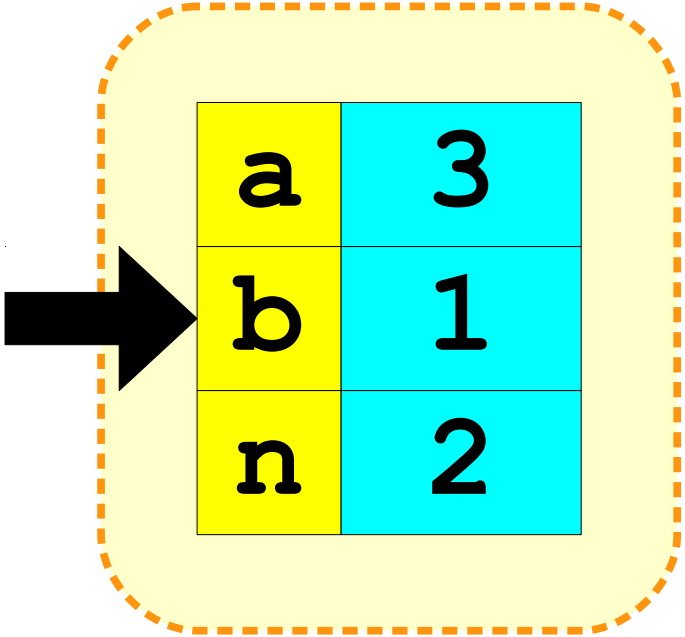


letterFreq

```
for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```

Counting Sort



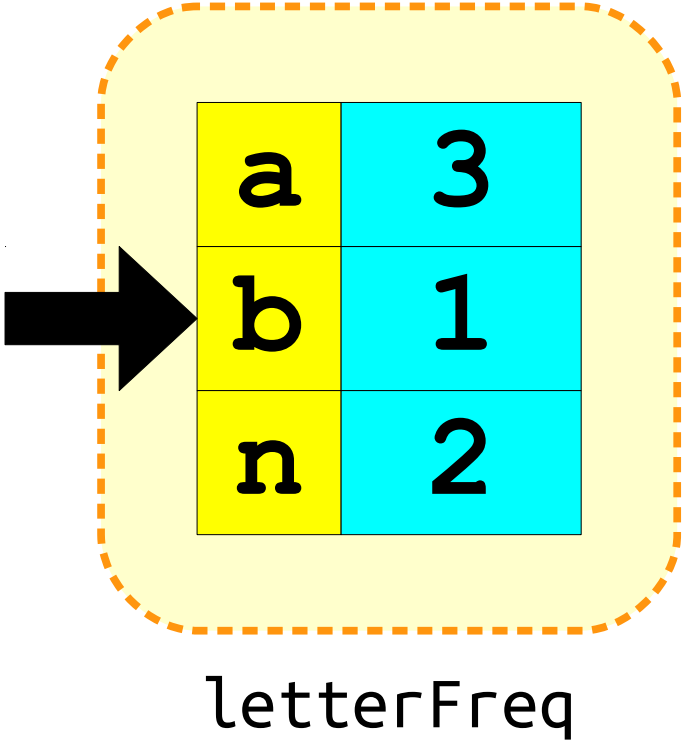


letterFreq

```
for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```

Counting Sort

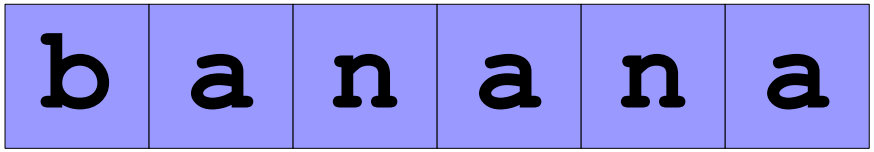
b	a	n	a	n	a

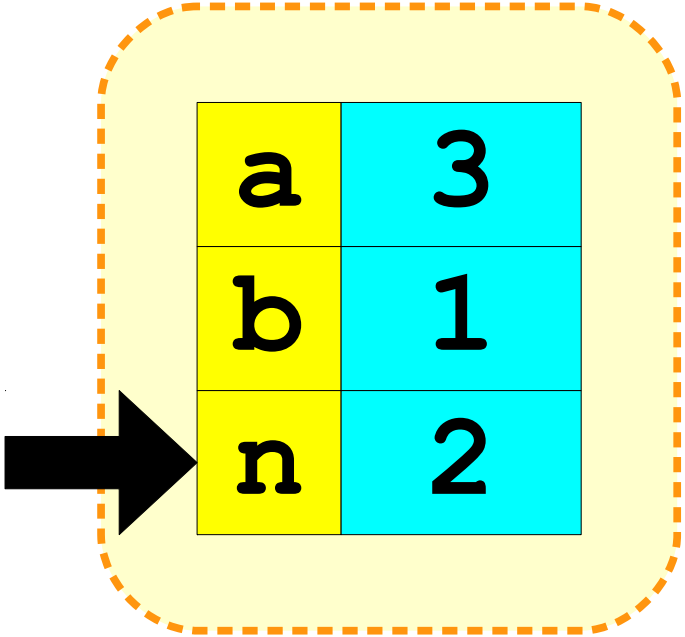



```
for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```

a	a	a	b

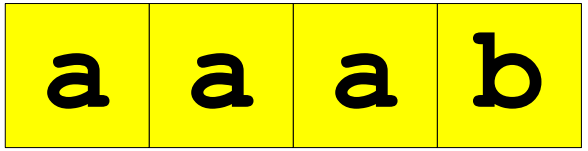
Counting Sort





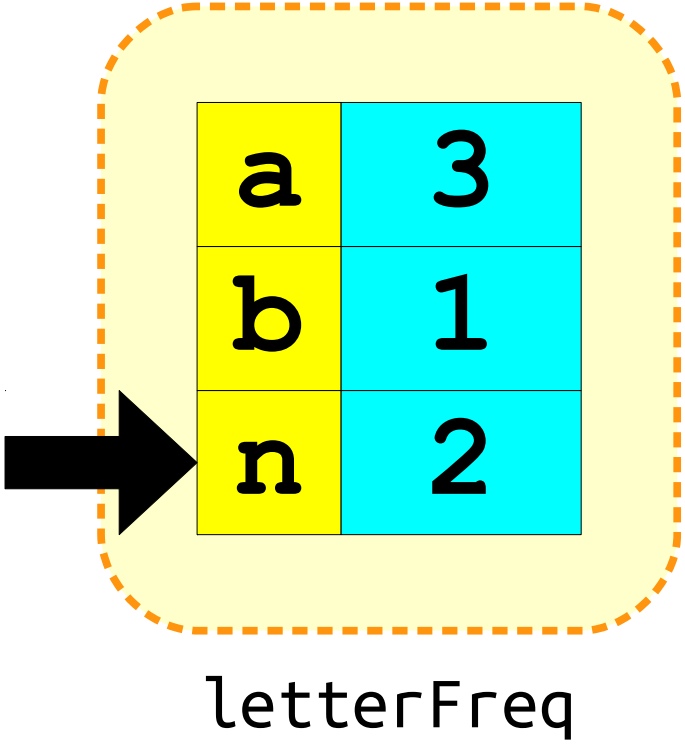
letterFreq

```
for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```



Counting Sort

b	a	n	a	n	a




```
for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```

a	a	a	b	n	n

Counting Sort

b	a	n	a	n	a

a	3
b	1
n	2

letterFreq

a	a	a	b	n	n