

# Array-Based Implementation

Tiziana Ligorio

# Today's Plan



Let's implement that Bag!!!

# Announcements

Sign up for the CUNY 2X Newsletter:

<http://eepurl.com/gkK8TD>

## Comments:

- Purpose: someone reading only the interface must be able to fully use your class without having to look at the implementation (like you do with `std::string`)
- No need to explain C++, anyone looking at your interface to use your class in their program should know the language (i.e. don't explain what include guards are in your comments)

# Announcements

Can we change the parameters of a function in the project?

# Announcements

Can we change the parameters of a function in the project?

**NO**

Why?

# Announcements

Can we change the parameters of a function in the project?

**NO**

Why?

Because you'd be writing a different program, not what is being requested

- Unit testing cannot correctly call your functions
- Your project manager would not be happy!

If you provide default arguments to undocumented parameters no one will ever know about them so no one will ever use them.

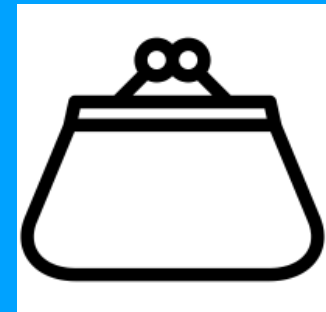
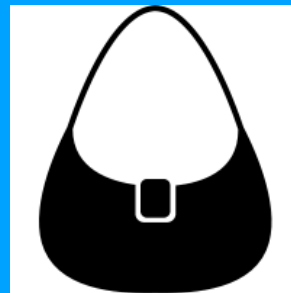
# Recap

An ADT is:

- A collection of data
- A set of operations on the data

Interface specifies **what** ADT operations do **not how**

# Bag



vector illustration eps10



# Implementation

# First step:

## Choose Data Structure

### So what is a Data Structure???

*A data organization and storage format that enables "efficient" access and modification.*

In this course we will encounter

Arrays

Vectors

Lists

Trees

Relative to the application  
You must choose the right  
data structure for your solution

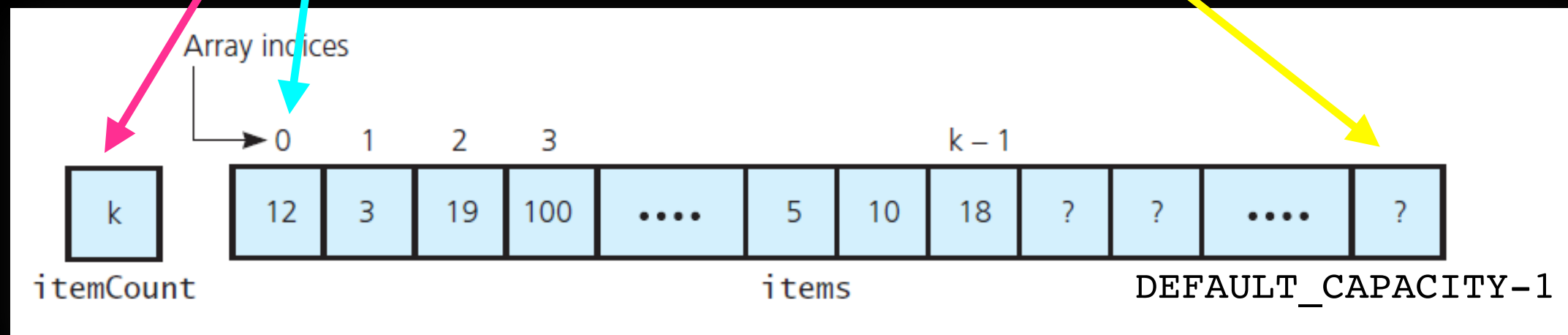
**ADT** defines the logical form  
**Data structure** is the physical implementation

# Array

A **fixed-size** container

**Direct access** to indexed location

Need to **keep track** of the number of elements in it



# ArrayBag

Name ArrayBag only for pedagogical purposes:

You would normally just call it a Bag and implement it as you wish

Because we will try different implementations, we are going to explicitly use the name of the data structure in the name of the ADT

Violates information hiding - wouldn't do it in "real life"

# Implementation Plan

Write the header file (`ArrayBag.hpp`) -> straightforward from design phase

**Incrementally** write/test implementation (`ArrayBag.cpp`)

Identify core methods / implement / test

- Create container (constructors)

- Add items

- Remove items...

E.g. you may want to add items before implementing and testing  
`getCurrentSize`

Use *stubs* when necessary

```
//STUB
```

```
int ArrayBag::getCurrentSize() const
{
    return 4; //STUB dummy value
}
```

# The Header File

```
#ifndef ARRAY_BAG_H_  
#define ARRAY_BAG_H_
```

**Include Guard:** used during linking to check that same header is not included multiple times.

```
#endif
```

# The Header File

```
#ifndef ARRAY_BAG_H_  
#define ARRAY_BAG_H_
```

```
#include "ArrayBag.cpp"  
#endif
```

**Include ArrayBag.cpp because this is a template.** Remember not to include the .cpp file in the project or compilation command



# The Header File

```
#ifndef ARRAY_BAG_H_
#define ARRAY_BAG_H_
```

```
template<class T>
class ArrayBag
```

```
}; //end ArrayBag
```

```
#include "ArrayBag.cpp"
#endif
```

**The class definition:**

define class ArrayBag as a **template**

Don't forget that *semicolon* at the end of your class definition!!!



# The Header File

```
#ifndef ARRAY_BAG_H_  
#define ARRAY_BAG_H_
```

```
template<class T>  
class ArrayBag  
{  
public:  
  
private:  
  
};    //end ArrayBag
```

```
#include "ArrayBag.cpp"  
#endif
```

**The public interface:** specifies the operations clients can call on objects of this class

**The private implementation:** specifies data and methods accessible only to members of this class. Invisible to clients

# The Header File

```
#ifndef ARRAY_BAG_H_
#define ARRAY_BAG_H_
```

```
template<class T>
class ArrayBag
{
```

public:

```
    ArrayBag();
    int getCurrentSize() const;
    bool isEmpty() const;
    bool add(const T& new_entry);
    bool remove(const T& an_entry);
    void clear();
    bool contains(const T& an_entry) const;
    int getFrequencyOf(const T& an_entry) const;
    std::vector<T> toVector() const;
```

This use of const means “I promise that this function doesn’t change the object”

This use of const means “I promise that this function doesn’t change the argument”

private:

```
};    //end ArrayBag
```

```
#include "ArrayBag.cpp"
#endif
```

**The public member functions** of the ArrayBag class. These can be called on objects of type ArrayBag  
Member functions are declared in the class definition. They will be implemented in the implementation file ArrayBag.cpp

# The Header File

```
#ifndef ARRAY_BAG_H_
#define ARRAY_BAG_H_
```

```
template<class T>
class ArrayBag
{
```

```
public:
    ArrayBag();
    int getCurrentSize() const;
    bool isEmpty() const;
    bool add(const T& new_entry);
    bool remove(const T& an_entry);
    void clear();
    bool contains(const T& an_entry) const;
    int getFrequencyOf(const T& an_entry) const;
    std::vector<T> toVector() const;

private:
    static const int DEFAULT_CAPACITY = 200 // Maximum Bag size
    T items_[DEFAULT_CAPACITY];             // Array of Bag items
    int item_count_;                         // Current count of Bag items
    /** @return index of target or -1 if target not found*/
    int get_index_of(const T& target) const;
}; //end ArrayBag
```

```
#include "ArrayBag.cpp"
#endif
```

**The private data members and helper functions** of the ArrayBag class. These can be called only within the ArrayBag implementation.

More than one public method will need to know the index of a target so we separate it out into a private helper function

# Implementation

```
#include "ArrayBag.hpp"
```

Include header: declaration of the methods this file implements

```
template<class T>
```

```
ArrayBag<T>::ArrayBag(): item_count_(0)
```

```
{  
} // end default constructor
```

Member Initializer List

```
#include "ArrayBag.hpp"
```

# Implementation

```
template<class T>
```

```
ArrayBag<T>::ArrayBag(): item_count_(0)
```

```
{  
} // end default constructor
```

```
template<class T>
```

```
int ArrayBag<T>::getCurrentSize() const
```

```
{  
    ???  
} // end getCurrentSize
```

```
template<class T>
```

```
bool ArrayBag<T>::isEmpty() const
```

```
{  
    ???  
} // end isEmpty
```

```
#include "ArrayBag.hpp"
```

# Implementation

```
template<class T>
```

```
ArrayBag<T>::ArrayBag(): item_count_(0)
{
    // end default constructor
}
```

```
template<class T>
```

```
int ArrayBag<T>::getCurrentSize() const
{
    return item_count_;
} // end getCurrentSize
```

```
template<class T>
```

```
bool ArrayBag<T>::isEmpty() const
{
    return item_count_ == 0;
} // end isEmpty
```

# Implementation

```
#include "ArrayBag.hpp"
```

```
• • •
```

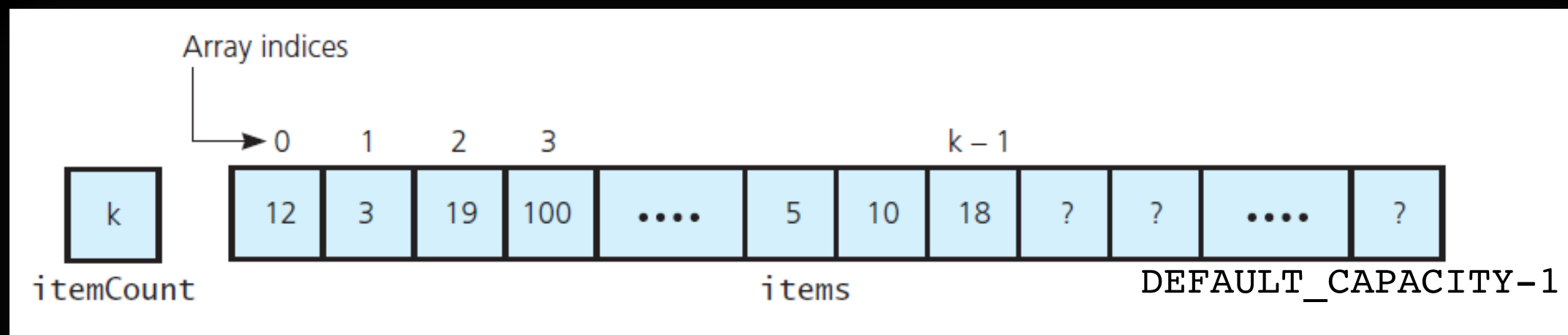
```
template<class T>
```

```
bool ArrayBag<T>::add(const T& new_entry)
```

```
{
```

```
    What do we need to do?
```

```
} // end add
```



# Implementation

```
#include "ArrayBag.hpp"
```

```
• • •
```

```
template<class T>
```

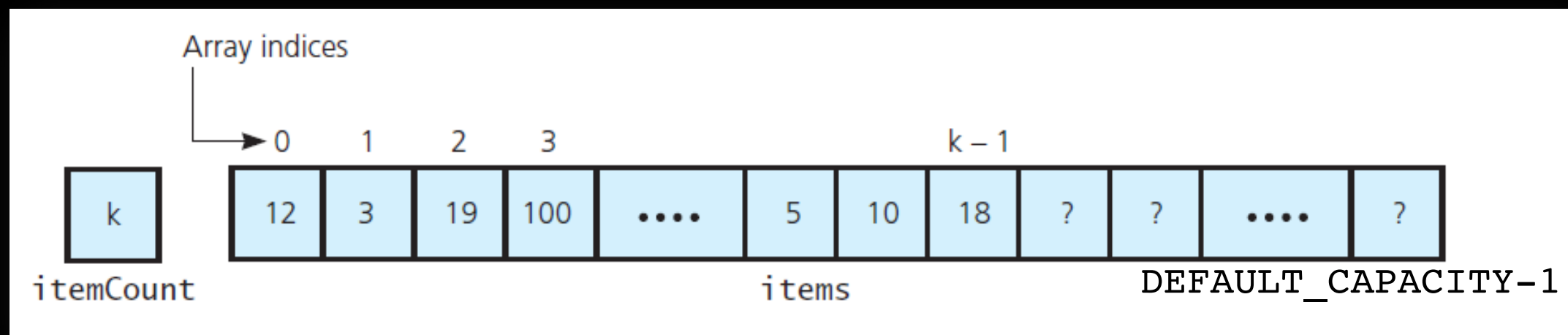
```
bool ArrayBag<T>::add(const T& new_entry)
```

```
{
```

```
    Check if there is room
```

```
    Add new_entry.. Where???
```

```
}    // end add
```





# Implementation

```
#include "ArrayBag.hpp"
```

```
• • •
```

```
template<class T>
```

```
bool ArrayBag<T>::add(const T& new_entry)  
{
```

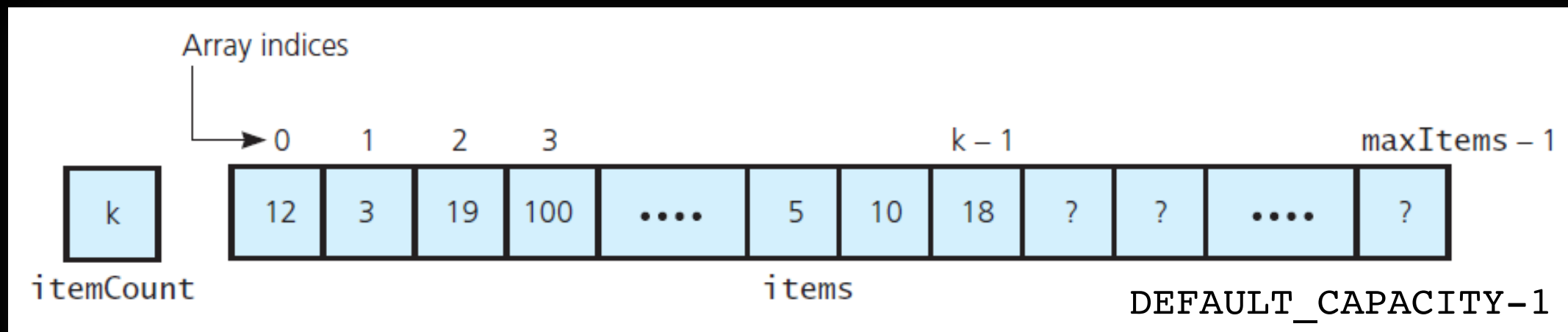
```
    Check if there is room
```

```
    Add new_entry.. At the end: index =
```

```
    item_count_
```

```
    Increment item_count_
```

```
} // end add
```



# Implementation

```
#include "ArrayBag.hpp"
```

```
...
```

```
template<class T>
```

```
bool ArrayBag<T>::add(const T& new_entry)
```

```
{
```

```
    bool has_room_to_add = (item_count_ <
                             DEFAULT_CAPACITY);
```

```
    if (has_room_to_add)
```

```
    {
```

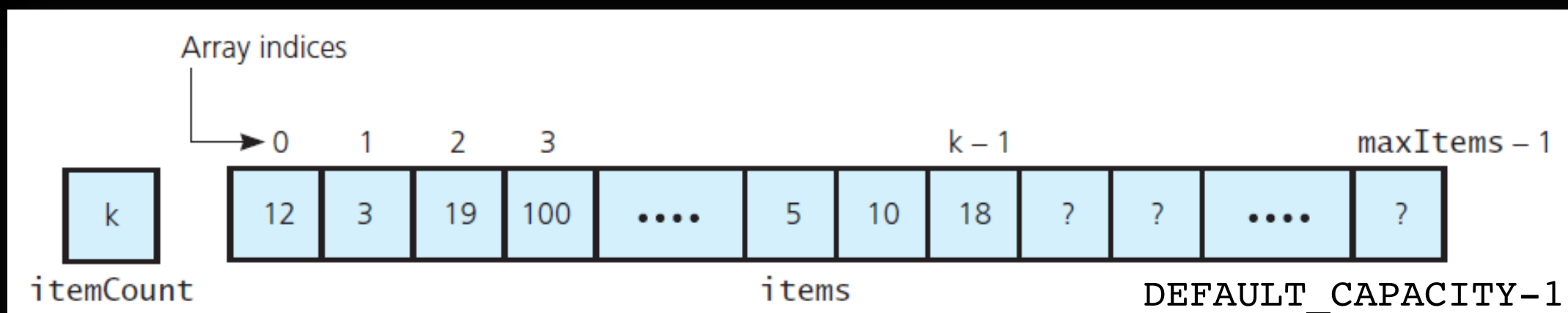
```
        items_[item_count_] = new_entry;
```

```
        item_count_++;
```

```
    } // end if
```

```
    return has_room_to_add;
```

```
} // end add
```



# Lecture Activity

Write Pseudocode for  
remove()

```
template<class T>
bool ArrayBag<T>::remove(const T& an_entry)
{
```

What do we need to do?

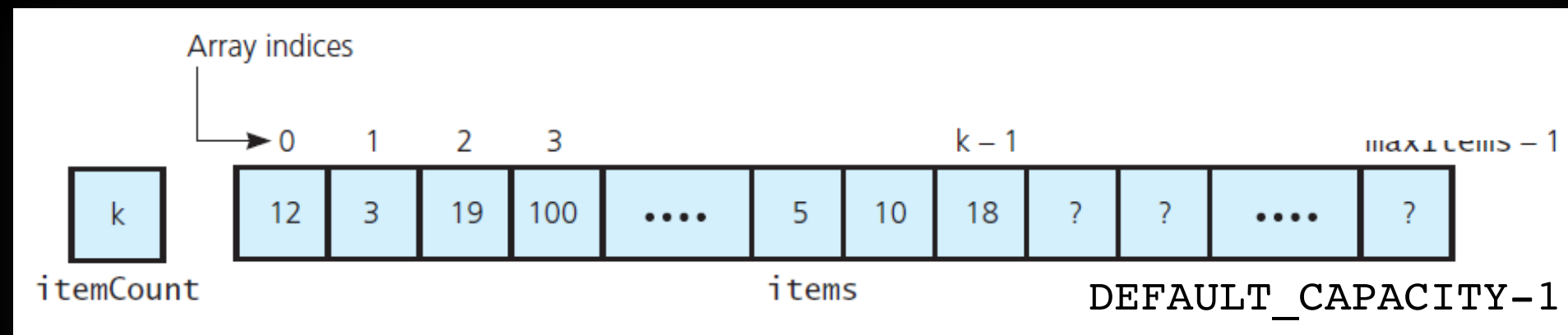
Hints:

- to add we looked if there was room in the bag. To remove what do we need to check first?

 Tricky

- we always strive for efficiency: think of how to remove with minimal "movement" / minimal number of operations and remember in a Bag ORDER DOES NOT MATTER

```
} //end remove
```



# Implementation

```
#include "ArrayBag.hpp"
```

```
...
```

```
template<class T>
```

```
bool ArrayBag<T>::remove(const T& an_entry)
```

```
{
```

```
    int located_index = getIndexOf(an_entry);
```

```
    bool can_remove_item = !isEmpty() && (located_index >= 0);
```

```
    if (can_remove_item)
```

```
    {
```

```
        item_count--;
```

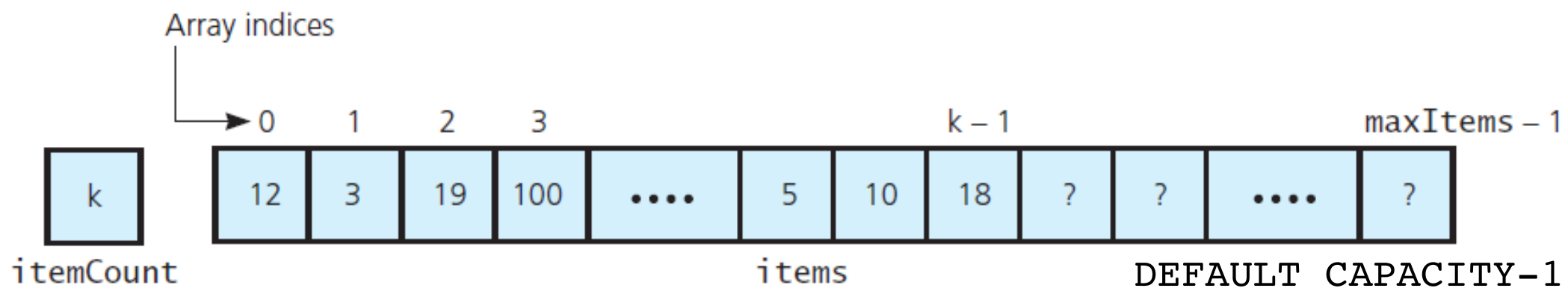
```
        items[located_index] = items[item_count]; //copy
```

```
        //last item in place of item to be removed
```

```
    } //end if
```

```
    return can_remove_item;
```

```
} //end remove
```



# Implementation

```
#include "ArrayBag.hpp"
```

```
...
```

```
template<class T>
```

```
bool ArrayBag<T>::remove(const T& an_entry)
```

```
{
```

```
    int located_index = getIndexOf(an_entry);
```

```
    bool can_remove_item = !isEmpty() && (located_index > -1)
```

```
    if (can_remove_item)
```

```
    {
```

```
        item_count--;
```

```
        items_[located_index] = items_[item_count_]; //copy
```

```
            //last item in place of item to be removed
```

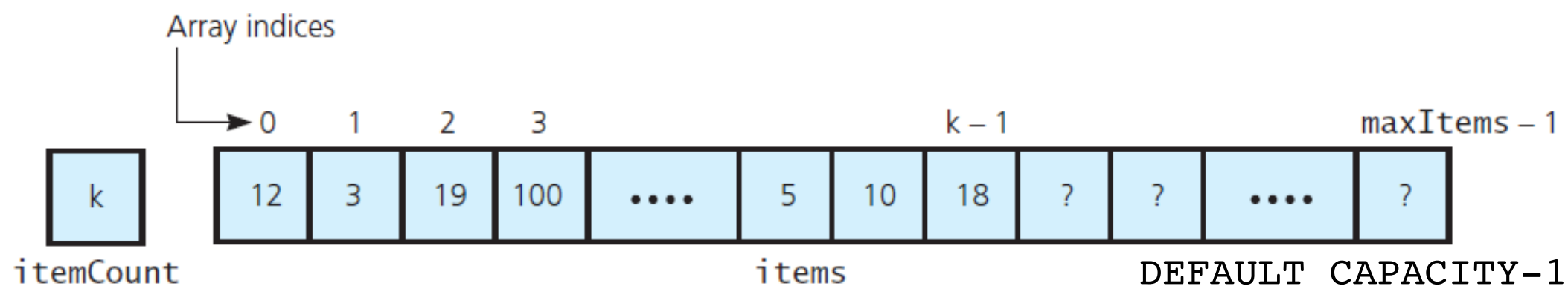
```
    } //end if
```

```
    return can_remove_item;
```

```
} //end remove
```

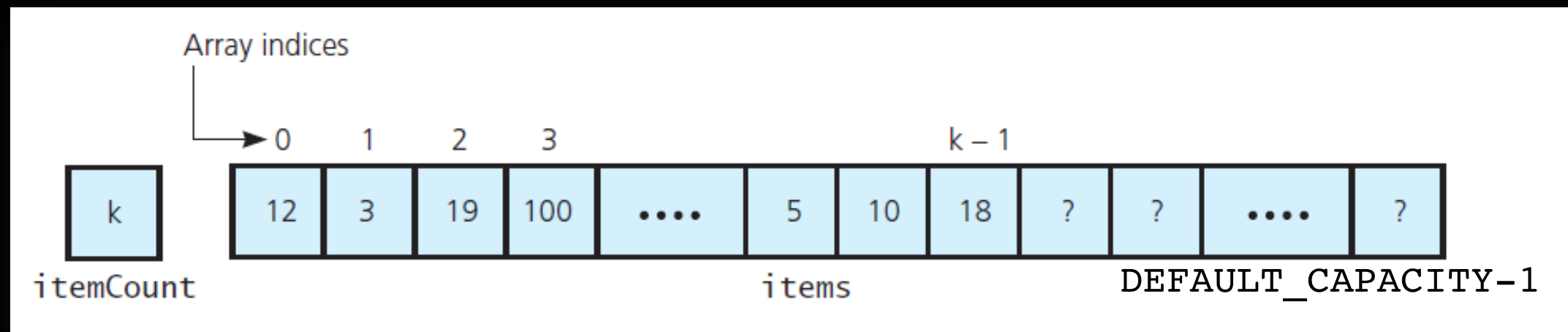
This is a messy Bag  
Order does not matter

What if we need  
to retain the order?



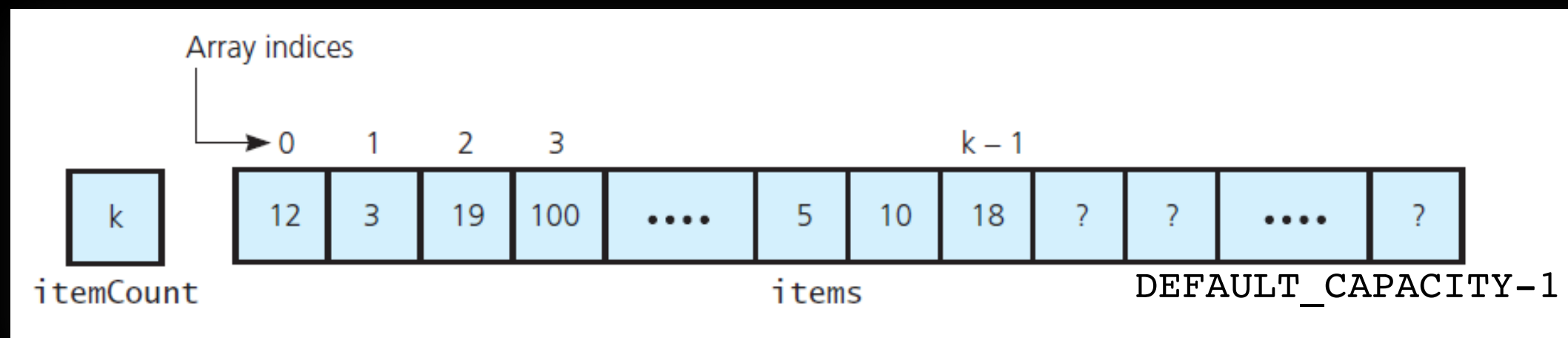
# Implementation

```
#include "ArrayBag.hpp"
template<class T>
int ArrayBag<T>::getFrequencyOf(const T&
                                an_entry) const
{
    What do we need to do???
} // end getFrequencyOf
```



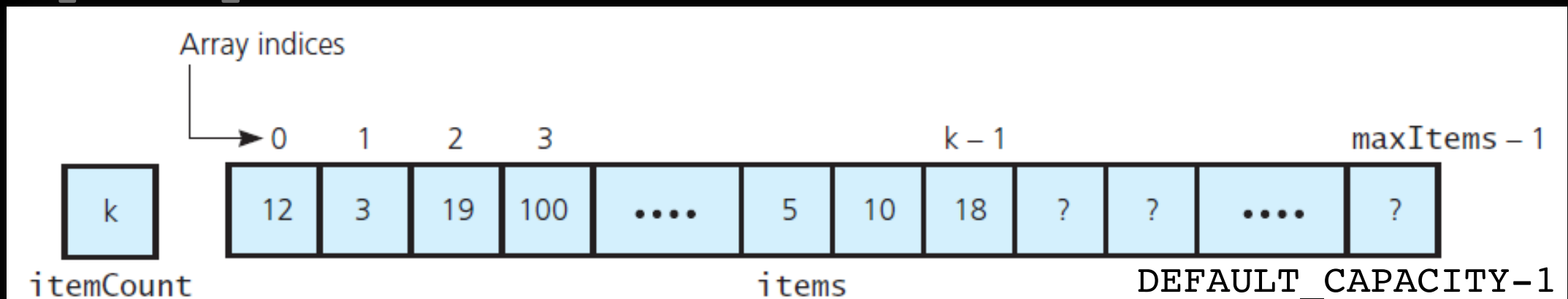
# Implementation

```
#include "ArrayBag.hpp"
template<class T>
int ArrayBag<T>::getFrequencyOf(const T&
                                an_entry) const
{
    Look at every array location,
    if == an_entry, count it!
} // end getFrequencyOf
```



# Implementation

```
#include "ArrayBag.hpp"
template<class T>
int ArrayBag<T>::getFrequencyOf(const T&
                                an_entry) const
{
    int frequency = 0;
    int current_index = 0; //array index currently
                           //being inspected
    while (current_index < item_count_)
    {
        if (items_[current_index] == an_entry)
        {
            frequency++;
        } // end if
        current_index++; //increment to next entry
    } //end while
    return frequency;
} //end getFrequencyOf
```



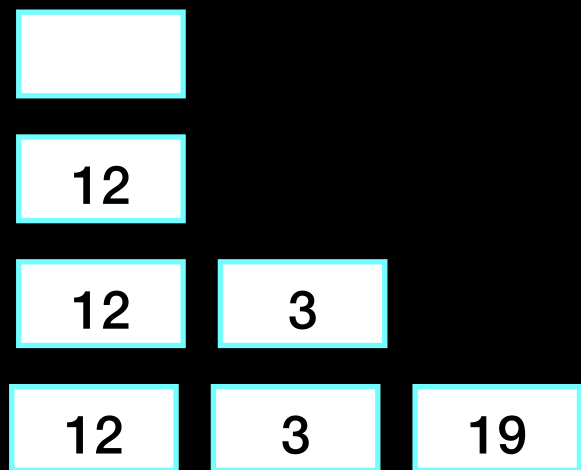
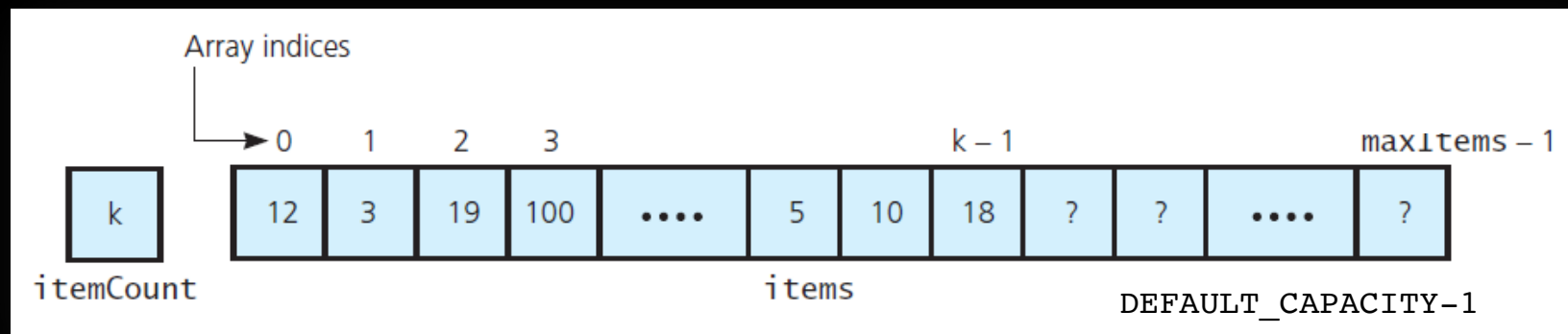


# Implementation

```
#include "ArrayBag.hpp"
template<class T>
std::vector<T> ArrayBag<T>::toVector() const
{
    std::vector<T> bag_contents;
    for (int i = 0; i < item_count_; i++)
        bag_contents.push_back(items_[i]);

    return bag_contents;
} // end toVector
```

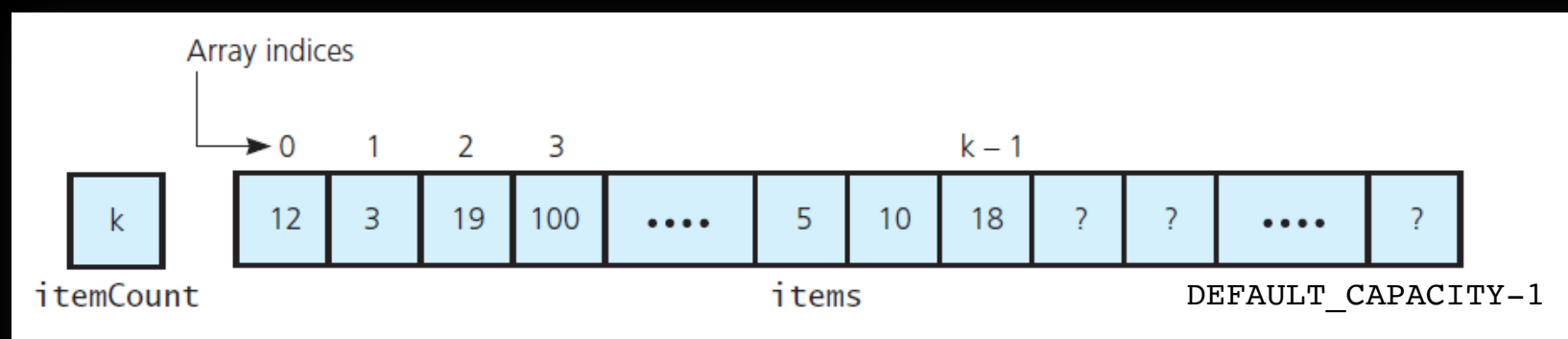
Return type



```
bag_contents.push_back(items_[0])
bag_contents.push_back(items_[1])
bag_contents.push_back(items_[2])
bag_contents.push_back(items_[3])
```

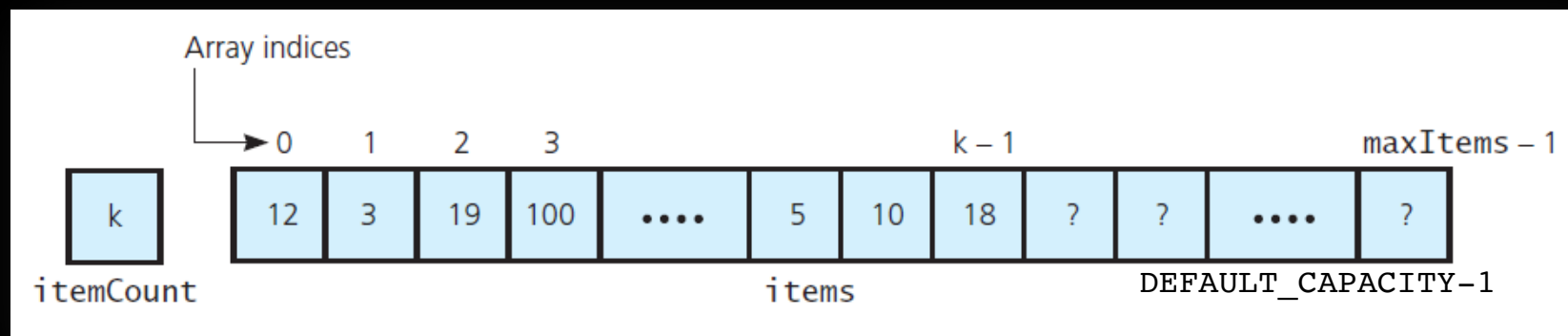
# Implementation

```
#include "ArrayBag.hpp"
// private
template<class T>
int ArrayBag<T>::getIndexOf(const T& target)
                                const
{
    Look at every array location,
    if == target return that location's index
} // end getIndexOf
```



# Implementation

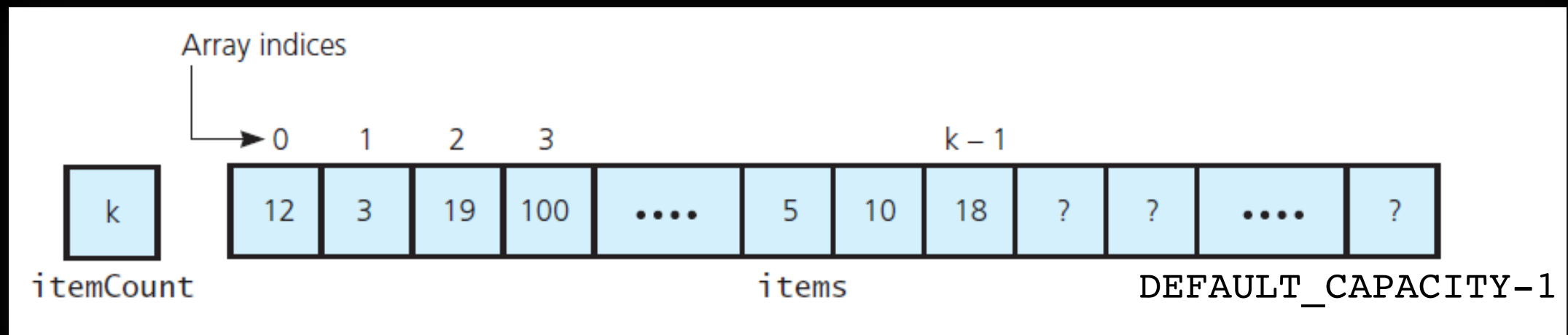
```
#include "ArrayBag.hpp"
template<class T> //private
int ArrayBag<T>::getIndexOf(const T& target) const {
    bool found = false;
    int result = -1;
    int search_index = 0;
    //If bag is empty item_count_ is zero loop is skipped
    while (!found && (search_index < item_count_)) {
        if (items[search_index] == target) {
            found = true;
            result = search_index;
        }
        else {
            search_index++;
        } //end if
    } //end while
    return result;
} //end getIndexOf
```



# Implementation

```
#include "ArrayBag.hpp"

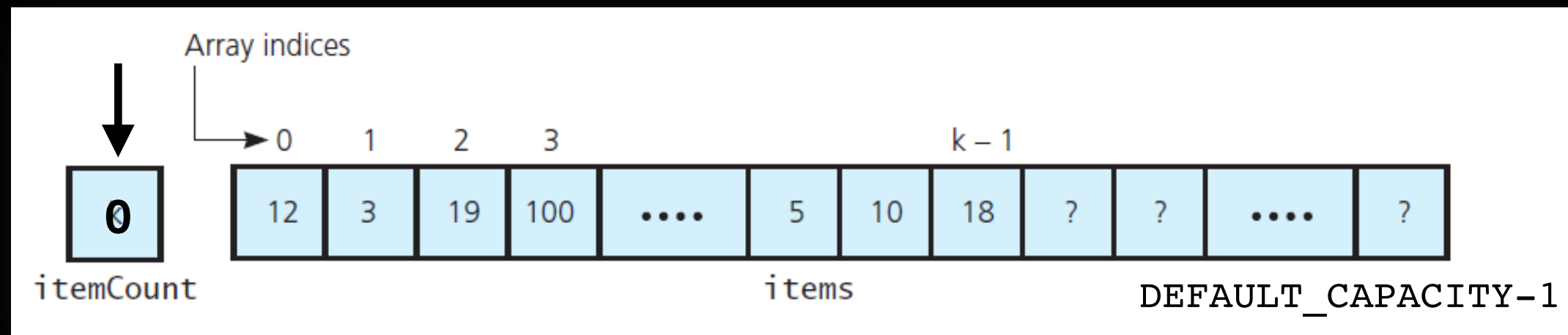
template<class T>
void ArrayBag<T>::clear()
{
    ???
} // end clear
```



# Implementation

```
#include "ArrayBag.hpp"

template<class T>
void ArrayBag<T>::clear()
{
    item_count_ = 0;
} // end clear
```



# Implementation

```
#include "ArrayBag.hpp"

template<class T>
bool ArrayBag<T>::contains(const T& an_entry)
                                const
{
    return getIndexOf(an_entry) > -1;
} // end contains
```

We have a working Bag!!!