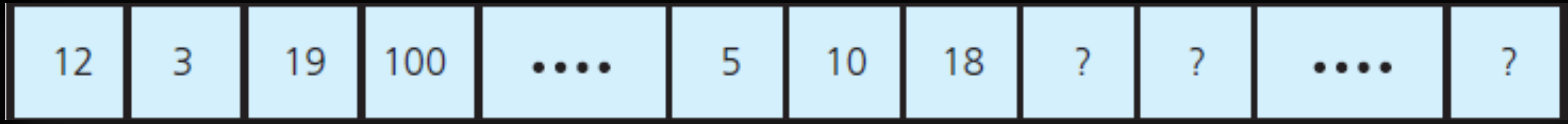
Array-Based Implementation



Tiziana Ligorio
Hunter College of The City University of New York

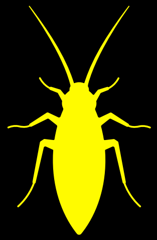
Announcements

- Interview prep workshops (stage 1 - online assessment):
    - 6 Meetings
    - Cub Hours: Tuesday 2:30-3:45
    - Start Tuesday February 13 (full schedule on Blackboard)
    - Incentive: 2pt Extra Credit (proportional to number of meetings attended)
    - Register at bit.ly/mtc_signup_S24

Project 2 opens today

Get started ASAP!!!

- Better finish early than stress out about a last minute bug!!!



How to get help

It is OK not to know something

It is NOT OK to do nothing about it!!!

Please ask for help, we are here to help you!!!

- Lab 1001B 11:30am-5:30pm

- Office hours Tuesdays and Fridays 11:30am-12:30pm or by appointment (email tligorio@hunter.cuny.edu)

- Ed Discussion: lot's of helpful posts OR ask a new question if it hasn't been asked already

Recap

We designed a Bag

ADT:

- A collection of data
- A set of operations on the data
- Specifies **what** **(interface)** ADT operations do, **not how**

Templates

- A place holder for type

Question

Did you implement and TEST MyTemplate?

Today’s Plan

Let’s implement that Bag!!!

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.

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

In this course we will encounter

- Arrays
- Vectors
- Lists
- Trees

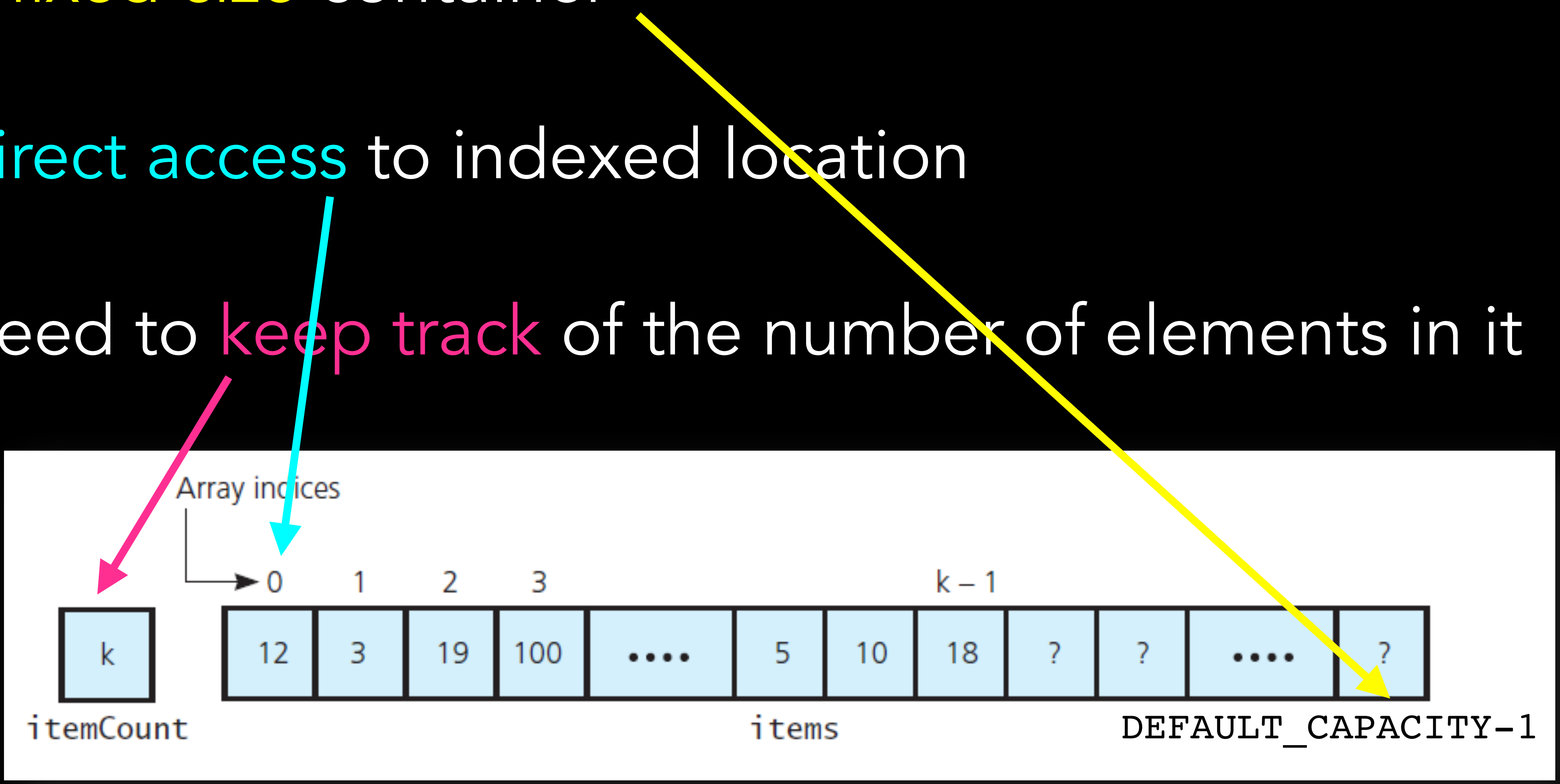
ADT defines the logical form
Data structure is the physical implementation

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 (.hpp)

```
#ifndef ARRAY_BAG_H_
#define ARRAY_BAG_H_









#endif
```

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

The Header File (.hpp)

```
#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 (.hpp)

```
#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 (.hpp)

```
#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 (.hpp)

```
#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:



};    //end ArrayBag

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

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"

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 (.hpp)

```
#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 getIndexOf(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 (.cpp)

```
#include "ArrayBag.hpp"


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

Include header: declaration of the methods this file implements

Member Initializer List

Implementation (.cpp)

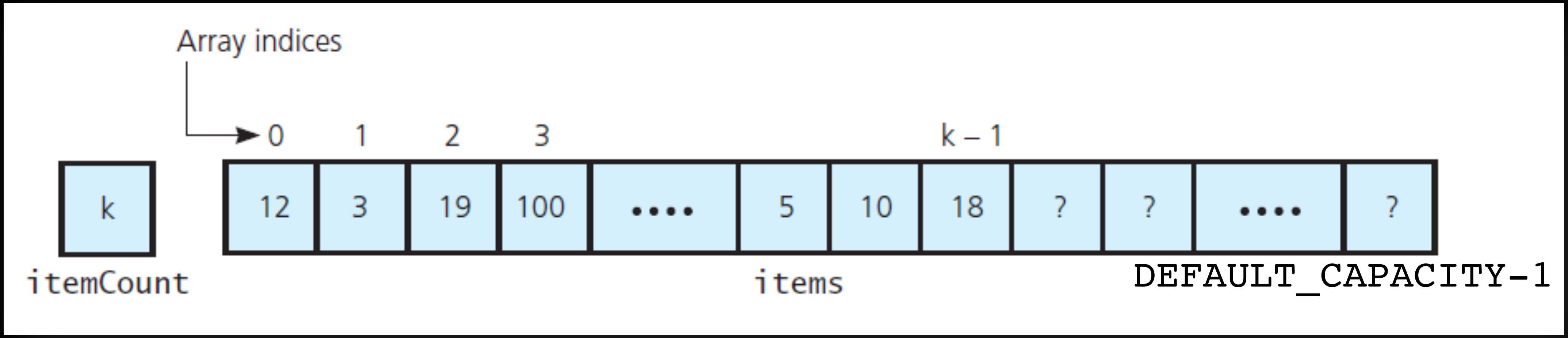
```
#include “ArrayBag.hpp”


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
```

Implementation (.cpp)

```
#include “ArrayBag.hpp”


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
```

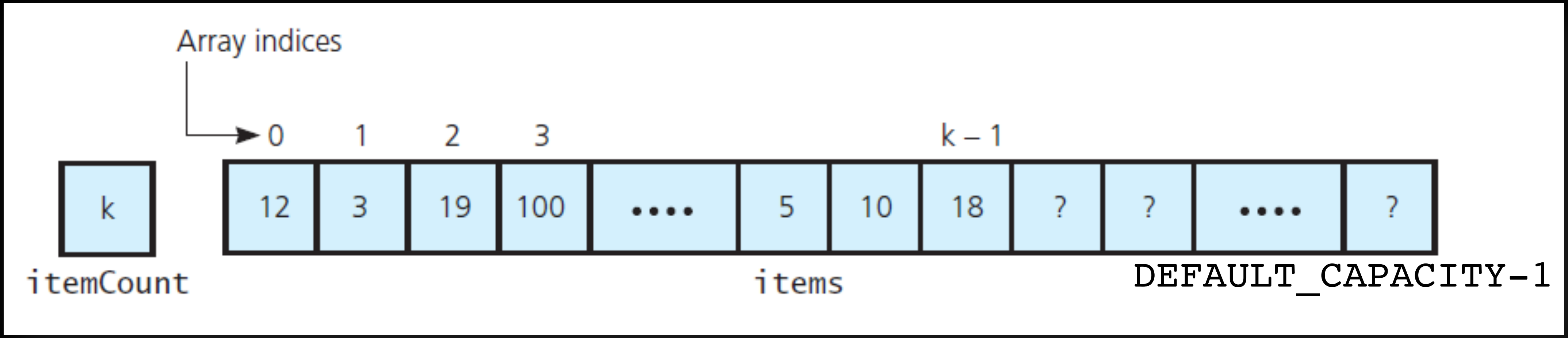
Implementation (.cpp)

```
#include “ArrayBag.hpp”

. . .


template<class T>
bool ArrayBag<T>::add(const T& new_entry)
{
    What do we need to do? (Hint: 2 things)
}  // end add
```



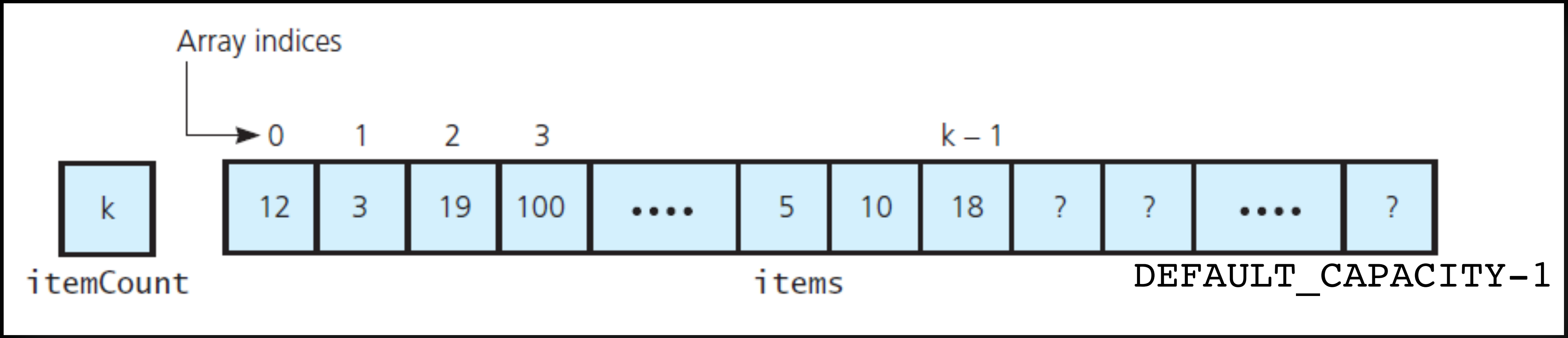
Implementation (.cpp)

```
#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
```



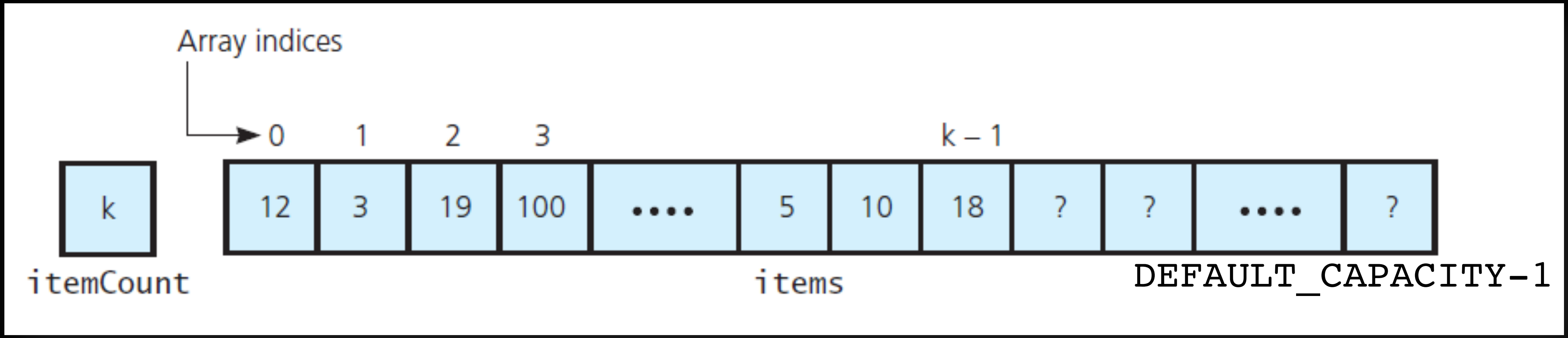
Implementation (.cpp)

```
#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 (.cpp)

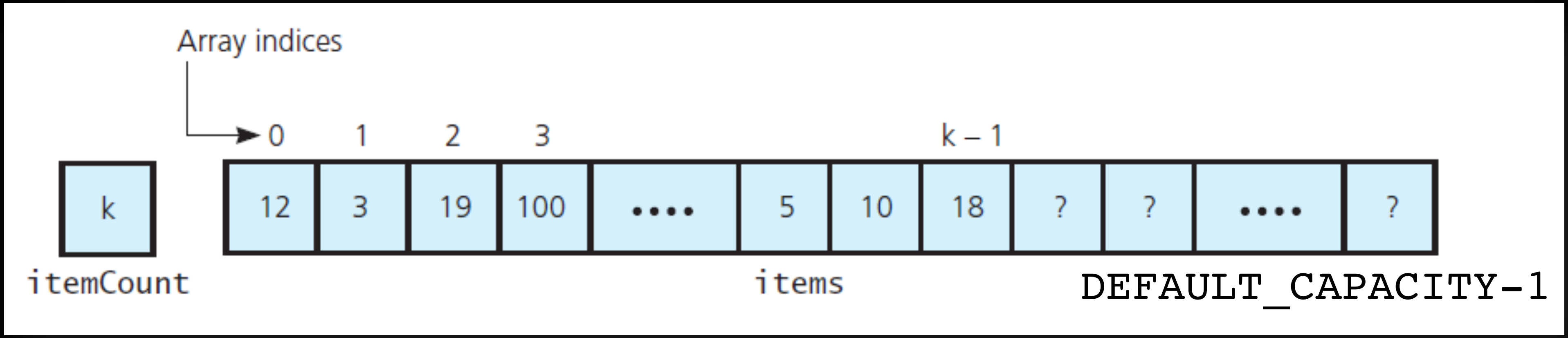
```
#include "ArrayBag.hpp"

. . .

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
```



Implementation (.cpp)

Write Pseudocode for remove()

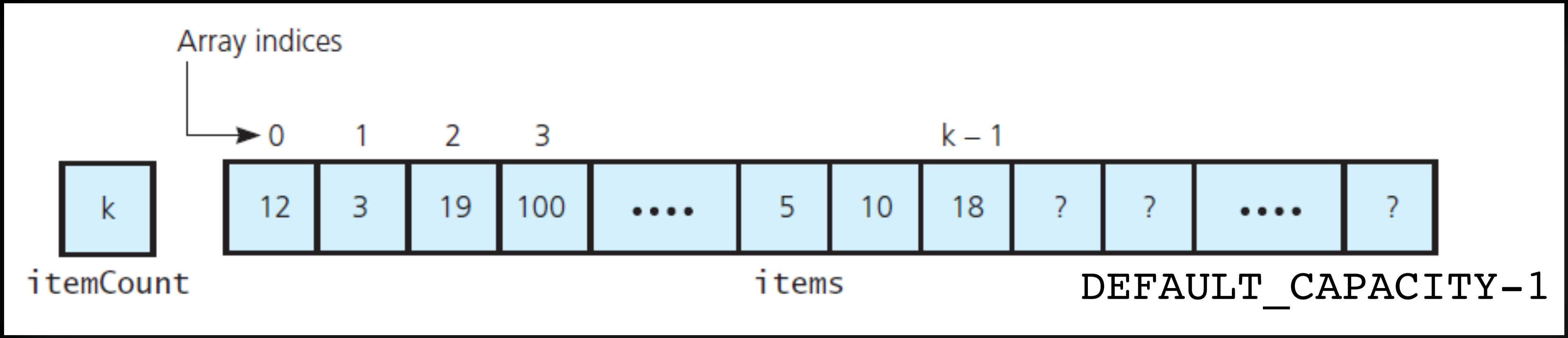
```
template<class T>
bool ArrayBag<T>::remove(const T& an_entry)
{

What do we need to do?

} //end remove
```



Implementation (.cpp)

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?

} //end remove
```

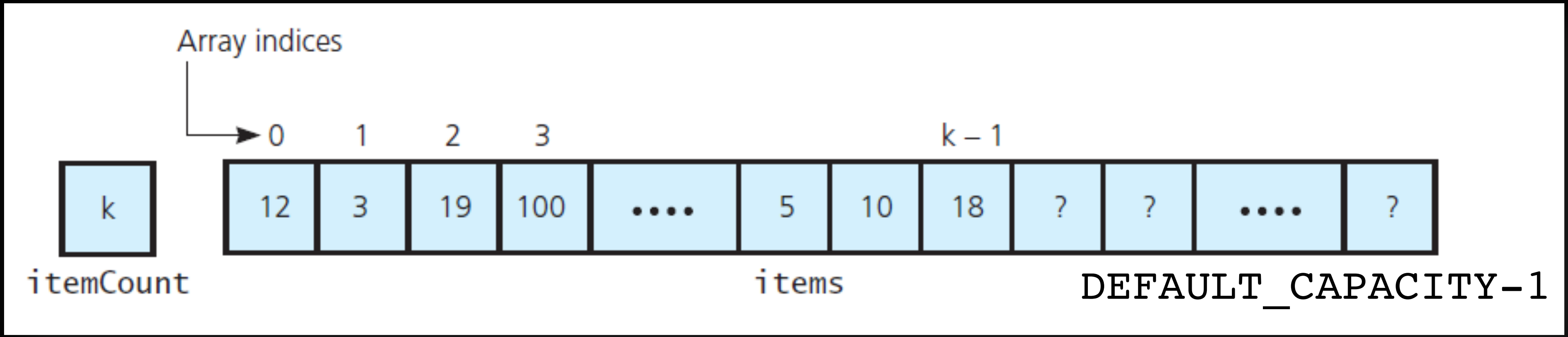


Implementation (.cpp)

Write Pseudocode for remove()

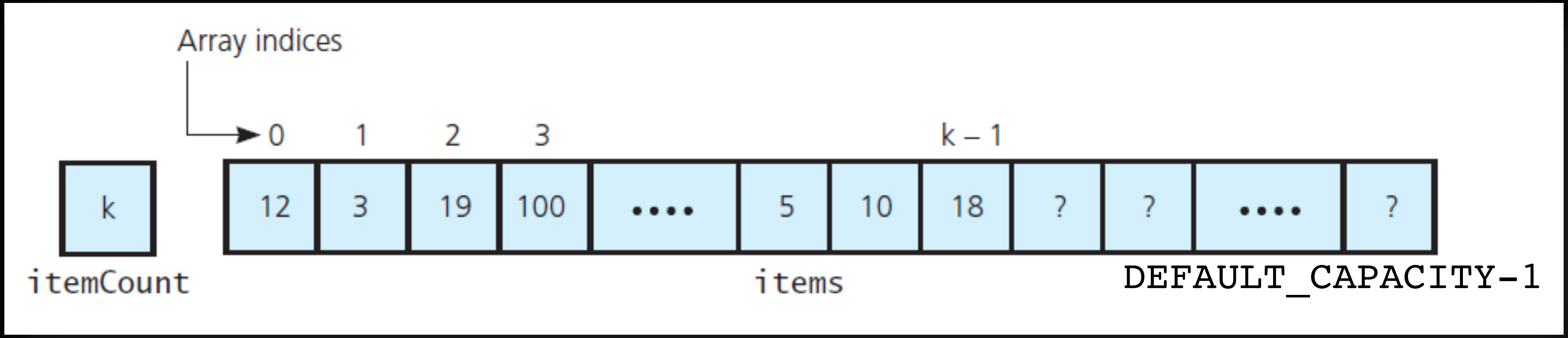
```
template<class T>
bool ArrayBag<T>::remove(const T& an_entry)
{

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?

- 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
```

Tricky 😏



Implementation (.cpp)

```
#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
```

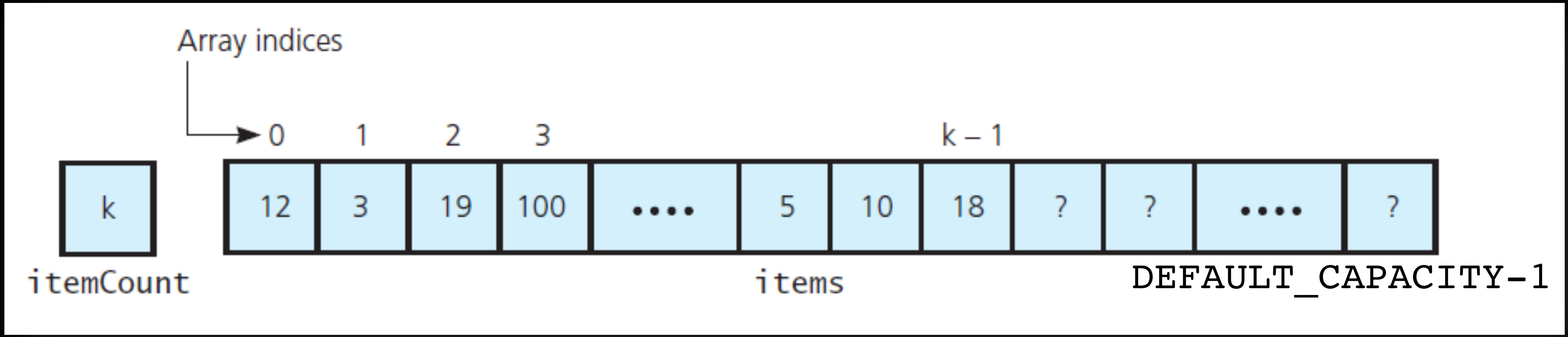


Implementation (.cpp)

```
#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
```

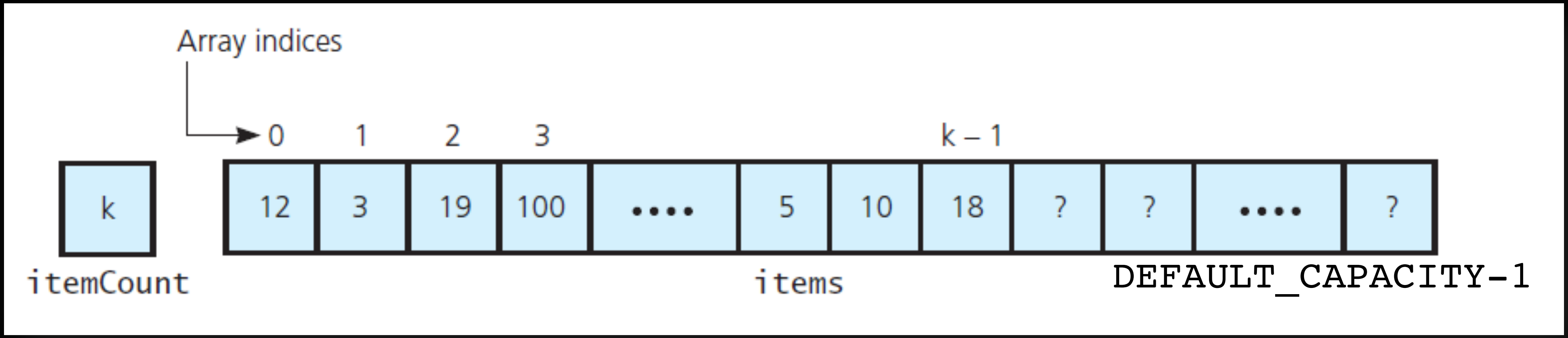
This is a messy Bag
Order does not matter

What if we need
to retain the order?



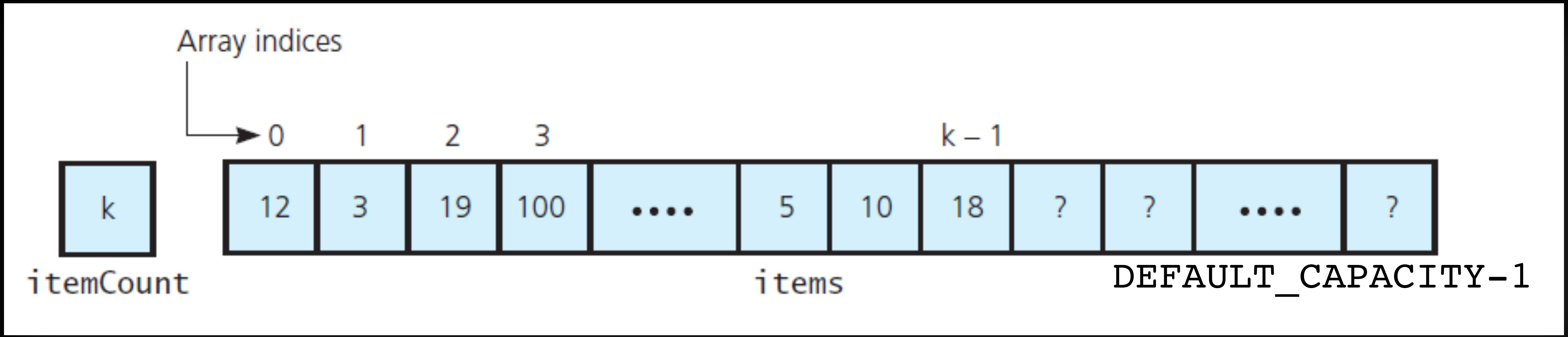
Implementation (.cpp)

```
#include “ArrayBag.hpp”


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



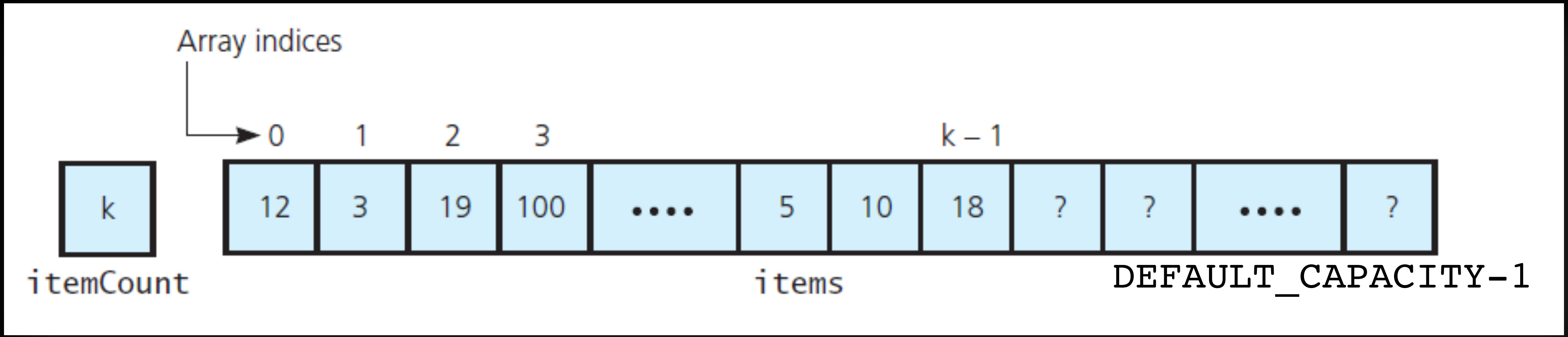
Implementation (.cpp)

```
#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 (.cpp)

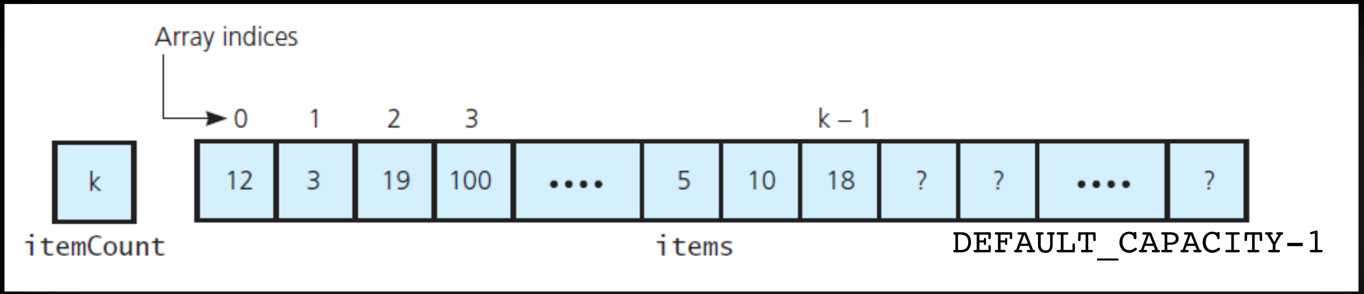
```
#include "ArrayBag.hpp"


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
```

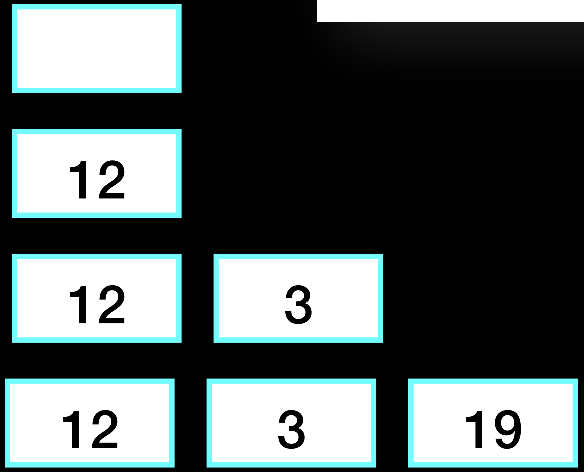


Implementation (.cpp)

```
#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

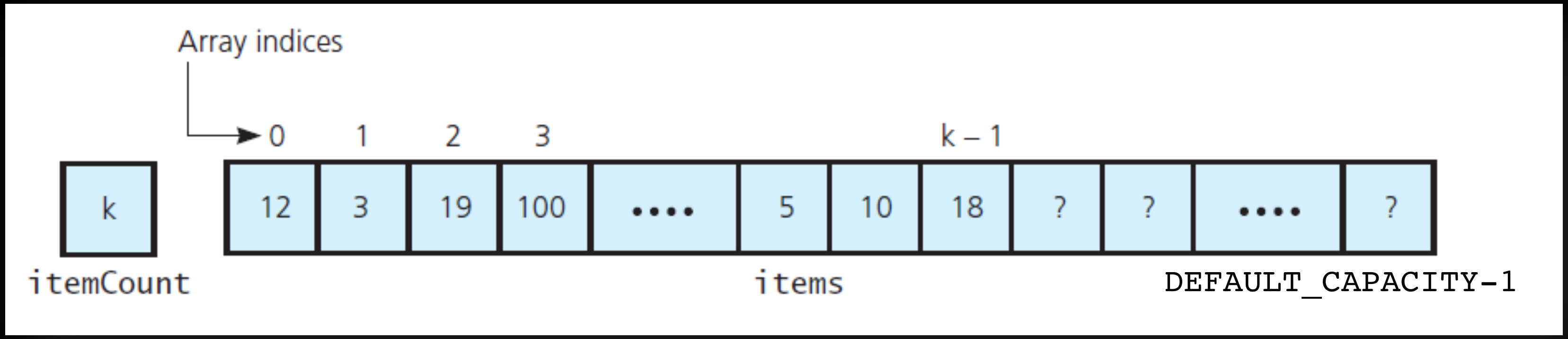




```
bag_contents.push_back(items_[0])

bag_contents.push_back(items_[1])

bag_contents.push_back(items_[2])

. . .
```

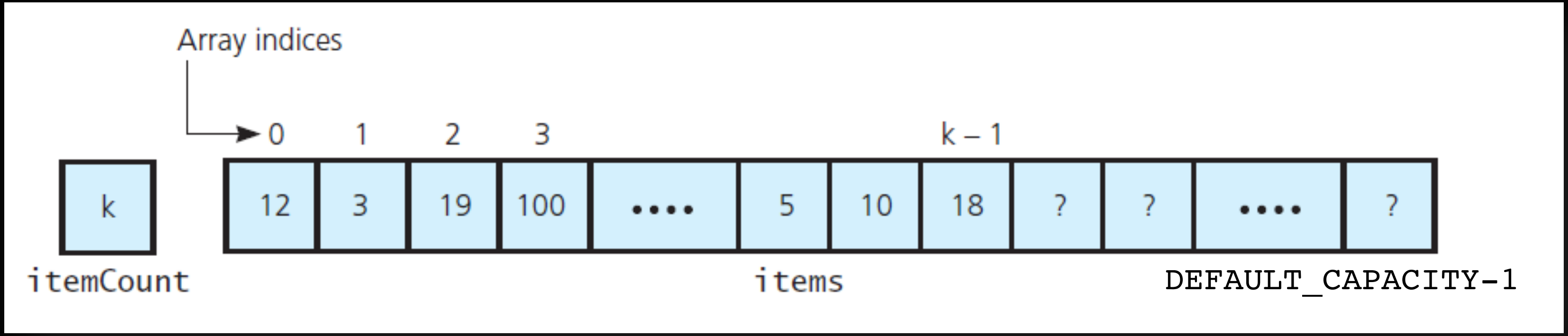
Implementation (.cpp)

```
#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
```



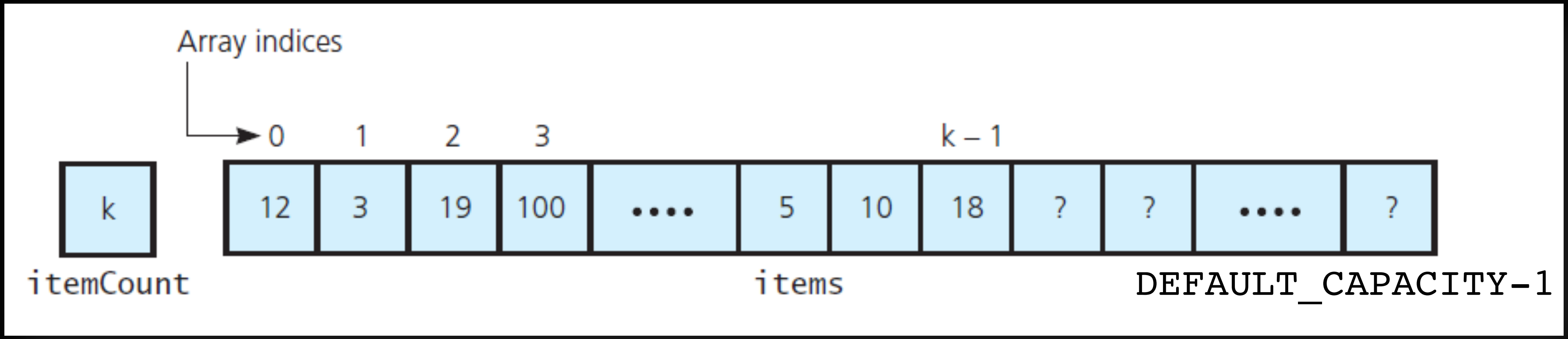
Implementation (.cpp)

```
#include “ArrayBag.hpp”


// private
template<class T>
int ArrayBag<T>::getIndexOf(const T& target) const
{
    bool found = false;
    int result = -1;
    int search_index = 0;

    // If the bag is empty, item_count_ is zero, so 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
```



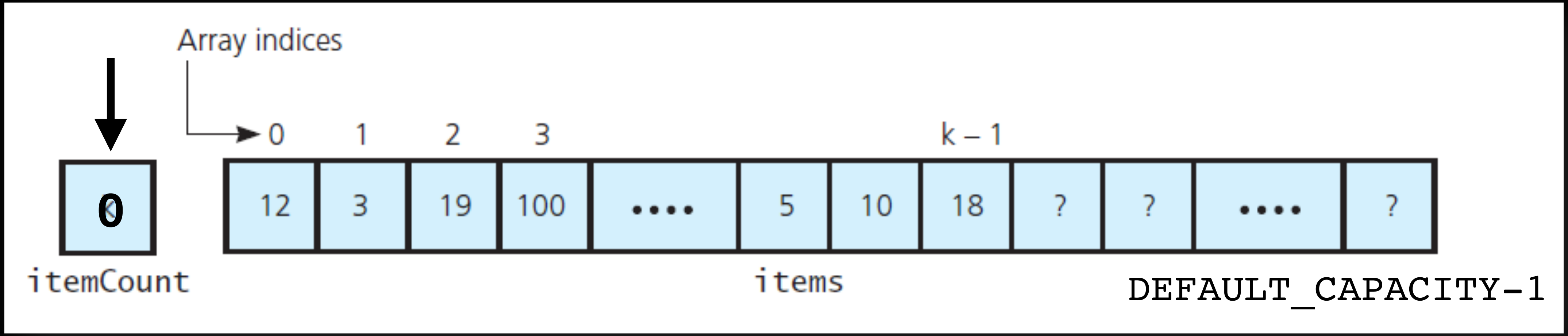
Implementation (.cpp)

```
#include “ArrayBag.hpp”



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



Implementation (.cpp)

```
#include "ArrayBag.hpp"



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



Implementation (.cpp)

```
#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!!!

Next time: Algorithm Efficiency