Array-Based Implementation

Tiziana Ligorio
tligorio@hunter.cuny.edu

Today’s Plan

Recap

Let’s implement that Bag!!!

Announcements and Syllabus Check

Running 1 lecture behind

Course webpage / Syllabus / Programming Rules: YOU MUST BE VERY FAMILIAR WITH THESE!!!

Skirball Science Learning Center: help on things you should know but don't
UTA's in lab: help with projects and exam prep

PLEASE feel free to ASK QUESTIONS during lecture!

ArrayBag files on Blackboard — Make sure you understand how every method works -> **exam questions directly based on projects**

Opportunities

JPMC jobs and internships through the Code for Good hackathon - Deadline September 19

Description: https://jpmchase.taleo.net/careersection/10140/jobdetail.ftl?job=180070311

Application: https://careers.jpmorgan.com/careers/US/en/programs/code-for-good

Palantir internship - Deadline September 23

https://jobs.lever.co/palantir/1a13a5e8-dc42-4655-a5de-dbc120763f1e

Recap

An ADT is:

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

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

Bag

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

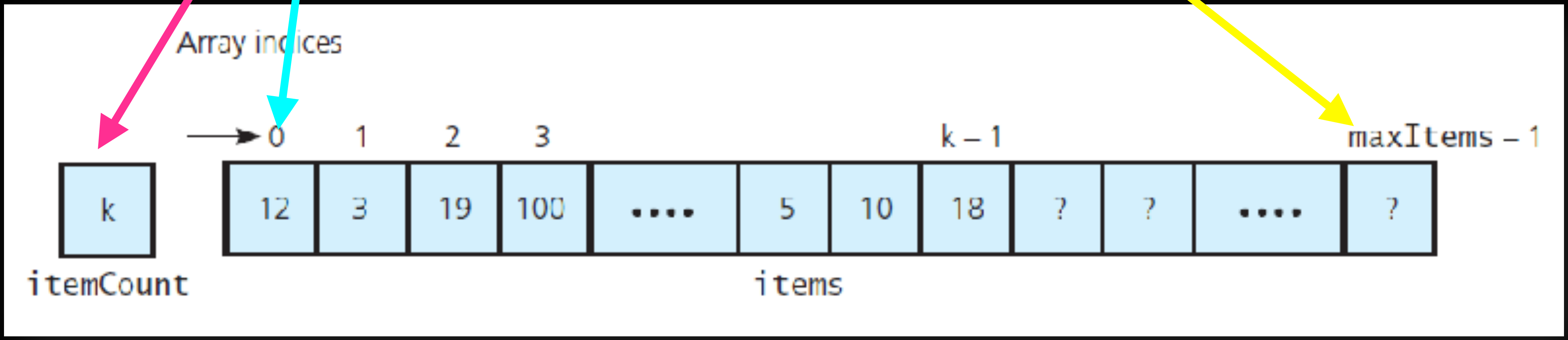
Arrays
Lists
Vectors
Stacks
Queues
Trees

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.h`) -> straightforward from design phase

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

Identify core methods / implement / test

- Create container (constructors)
- Add items
- Remove items…

We will look at these sequentially following the interface

When coding and testing you may want to `add` items before implementing and testing `getCurrentSize`

Use *stubs* when necessary

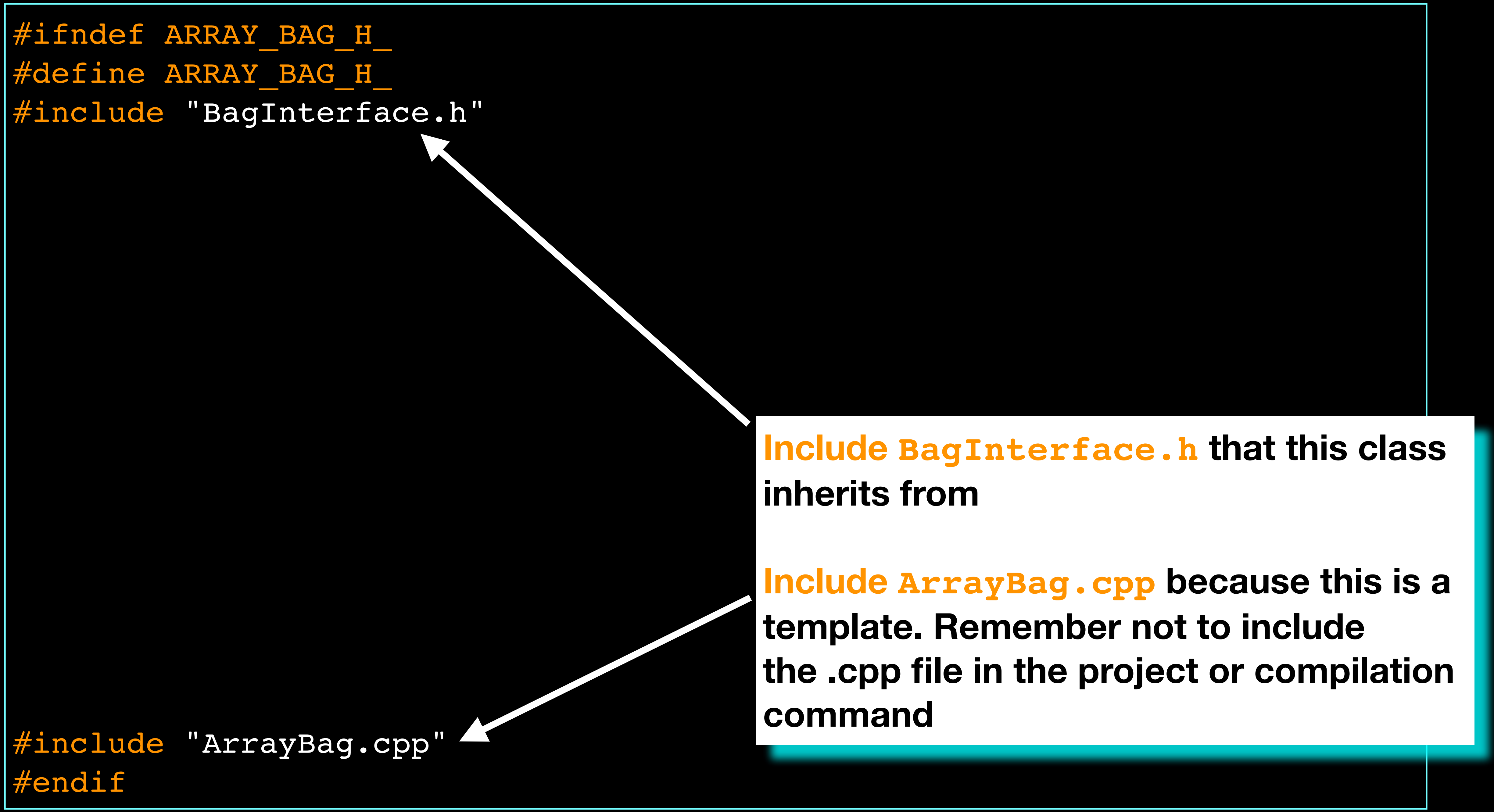
```
//STUB

bool add(const ItemType& newEntry)
{
    return true; //STUB
}
```

The Header File

```
#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



```
#ifndef ARRAY_BAG_H_
#define ARRAY_BAG_H_
#include "BagInterface.h"











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

Include `BagInterface.h` that this class inherits from

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_
#include “BagInterface.h”

template<class ItemType>
class ArrayBag : public BagInterface<ItemType>
{






};      //end ArrayBag

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

The class definition:
define class `ArrayBag` as a template
Inherit from `BagInterface`

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

The Header File

```
#ifndef ARRAY_BAG_H_
#define ARRAY_BAG_H_
#include “BagInterface.h”

template<class ItemType>
class ArrayBag : public BagInterface<ItemType>
{

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_
#include “BagInterface.h”

template<class ItemType>
class ArrayBag : public BagInterface<ItemType>
{

public:
    ArrayBag();
    int getCurrentSize() const;
    bool isEmpty() const;
    bool add(const ItemType& newEntry);
    bool remove(const ItemType& anEntry);
    void clear();
    bool contains(const ItemType& anEntry) const;
    int getFrequencyOf(const ItemType& anEntry) const;
    std::vector<ItemType> toVector() const;

private:



};     //end ArrayBag

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

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

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_
#include “BagInterface.h”

template<class ItemType>
class ArrayBag : public BagInterface<ItemType>
{

public:
     ArrayBag();
     int getCurrentSize() const;
     bool isEmpty() const;
     bool add(const ItemType& newEntry);
     bool remove(const ItemType& anEntry);
     void clear();
     bool contains(const ItemType& anEntry) const;
     int getFrequencyOf(const ItemType& anEntry) const;
     std::vector<ItemType> toVector() const;

private:
     static const int DEFAULT_CAPACITY = 6  // Small size to test for a full bag
     ItemType items_[DEFAULT_CAPACITY];      // Array of bag items
     int item_count_;                         // Current count of bag items
     int max_items_;                          // Max capacity of the bag
     // return: index of target or -1 if target not found
     int get_index_of_(const ItemType& target) const;
};     //end ArrayBag

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

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

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

Implementation

```
#include “ArrayBag.h”


template<class ItemType>
ArrayBag<ItemType>::ArrayBag(): item_count_(0), max_items_(DEFAULT_CAPACITY)
{
}  // end default constructor
```

Include header: declaration of the methods this file implements

Member Initializer List

Implementation

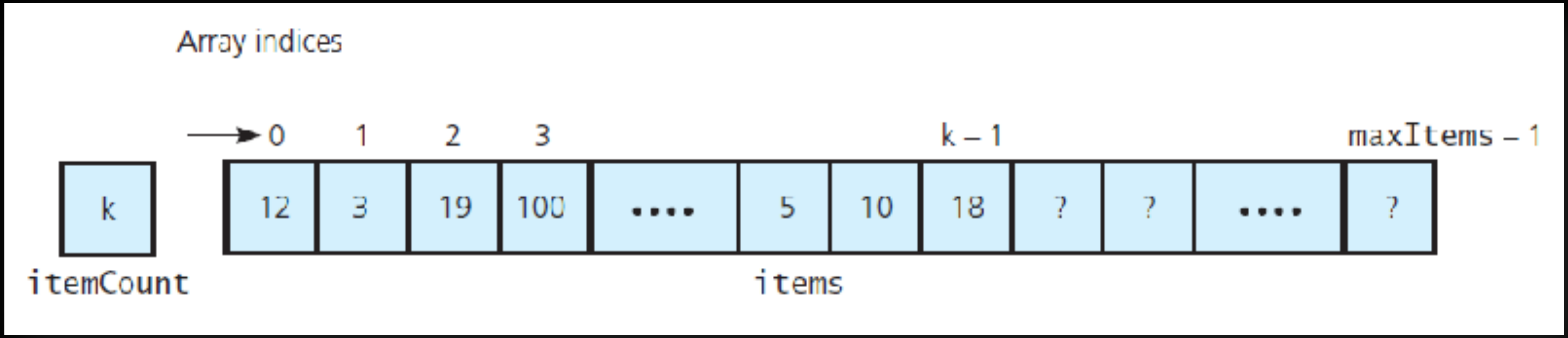
```
#include “ArrayBag.h”


template<class ItemType>
ArrayBag<ItemType>::ArrayBag(): item_count_(0), max_items_(DEFAULT_CAPACITY)
{
}  // end default constructor


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

Implementation

```
#include “ArrayBag.h”


template<class ItemType>
ArrayBag<ItemType>::ArrayBag(): item_count_(0), max_items_(DEFAULT_CAPACITY)
{
}  // end default constructor


template<class ItemType>
int ArrayBag<ItemType>::getCurrentSize() const
{
    return item_count_;
}  // end getCurrentSize


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

Implementation

```
#include “ArrayBag.h”

. . .

template<class ItemType>
bool ArrayBag<ItemType>::add(const ItemType& newEntry)
{
    bool has_room_to_add = (item_count_ < max_items_);
    if (has_room_to_add)
    {
        items_[item_count_] = newEntry;
        item_count_++;
    }  // end if
    return has_room_to_add;
}  // end add
```

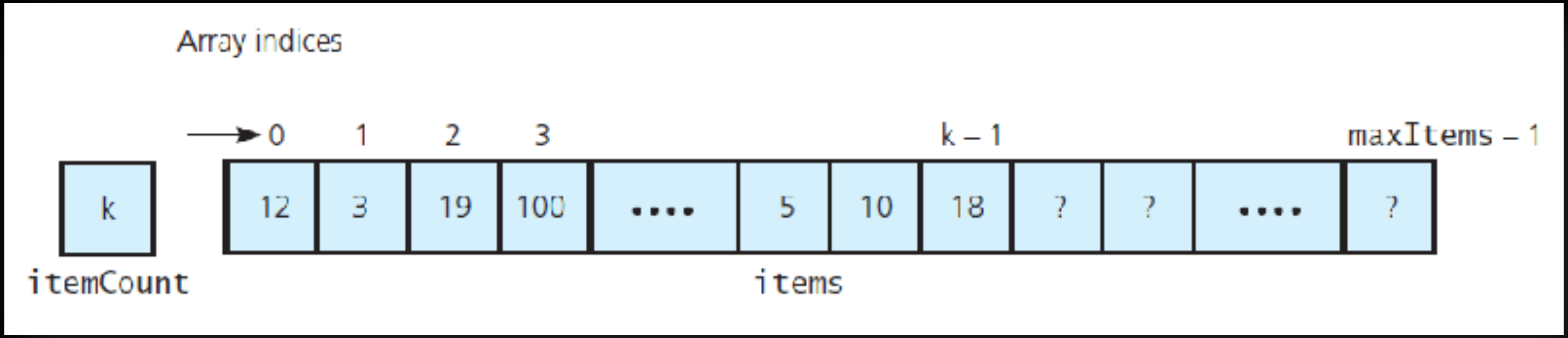


In-Class Task

```
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& anEntry)
{









} //end remove
```

Implementation

```
#include “ArrayBag.h”

. . .

template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& anEntry)
{
   int located_index = getIndexOf(anEntry);
    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
```



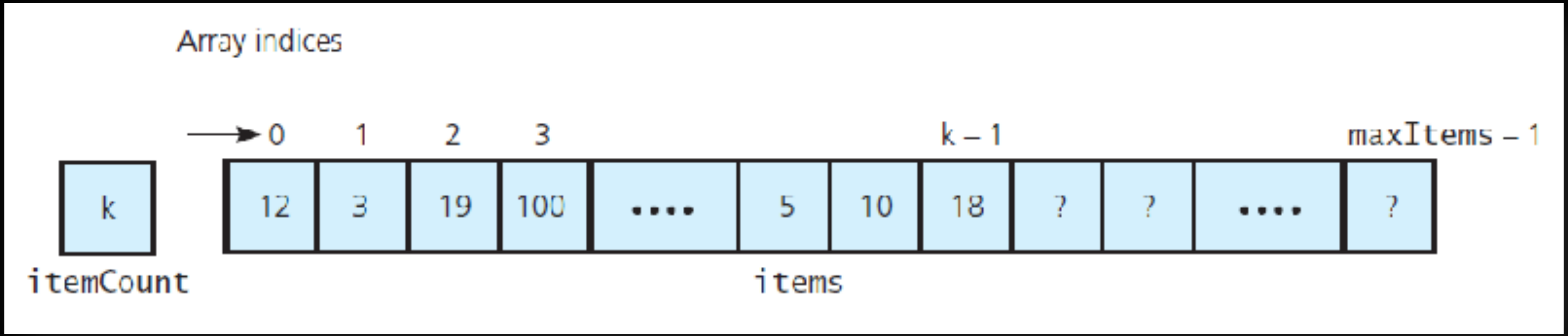
Implementation

```
#include "ArrayBag.h"

. . .

template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& anEntry)
{
   int located_index = getIndexOf(anEntry);
    bool can_remove_item = !isEmpty() && (located_index > -1);
    if (can_remove_item)
    {
        item_count_--;
        items_[located_index] = items_[item_count_];
    }   // end if
    return can_remove_item;
}  // end remove
```

This is a messy Bag
Order does not matter

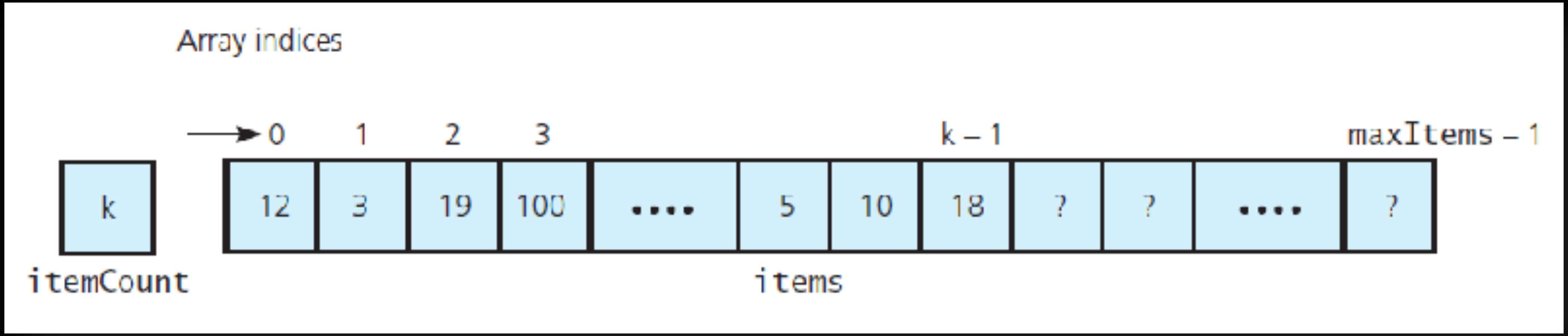
What were we doing
for GeniusBar?

What if we need
to retain the order?



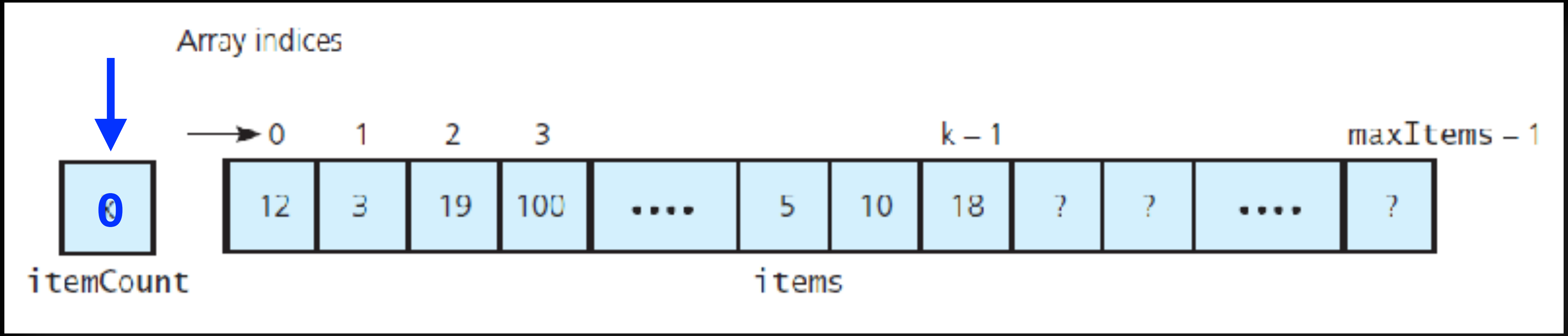
Implementation

```
#include “ArrayBag.h”



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



Implementation

```
#include “ArrayBag.h”



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

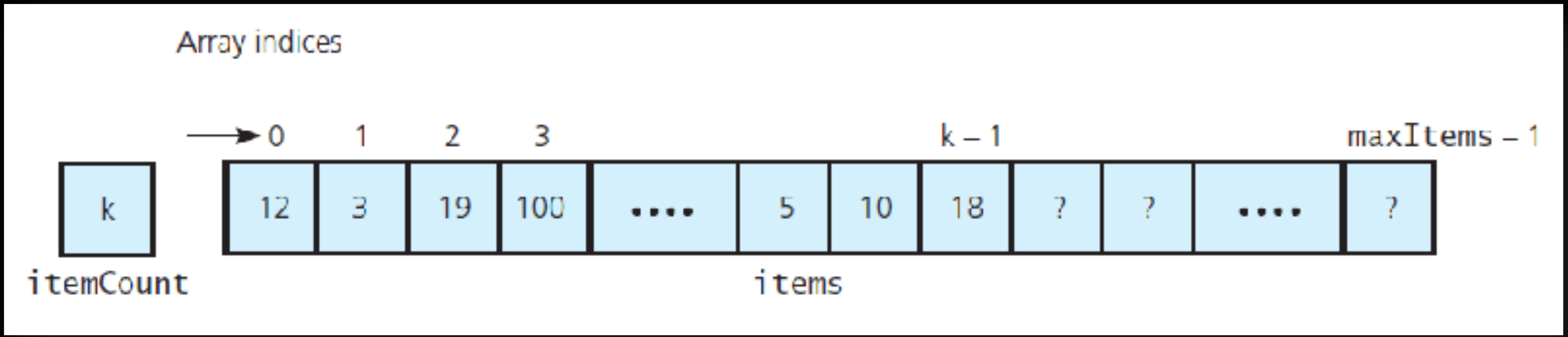


Implementation

```
#include “ArrayBag.h”



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

Implementation

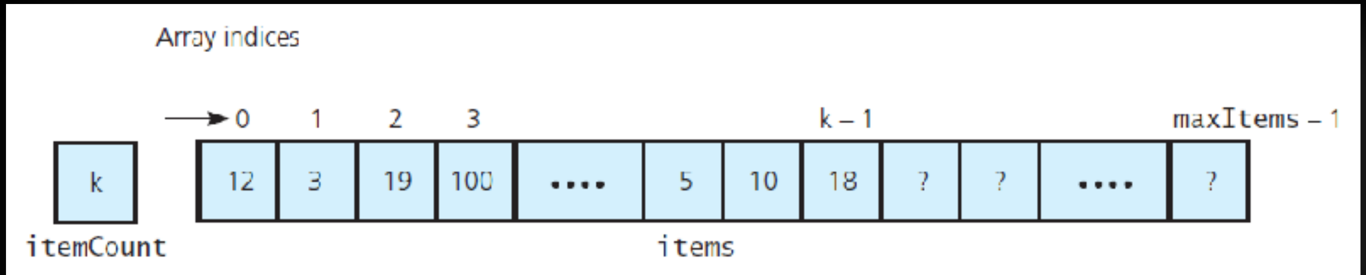
```
#include “ArrayBag.h”


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

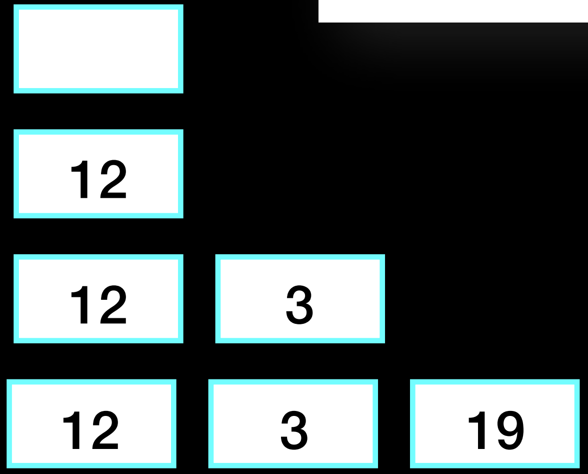


Implementation

```
#include “ArrayBag.h”


template<class ItemType>
std::vector<ItemType> ArrayBag<ItemType>::toVector() const
{
    std::vector<ItemType> bag_contents;
    for (int i = 0; i < itemCount; i++)
       bag_contents.push_back(items_[i]);

   return bag_contents;
}  // end toVector
```

Return type

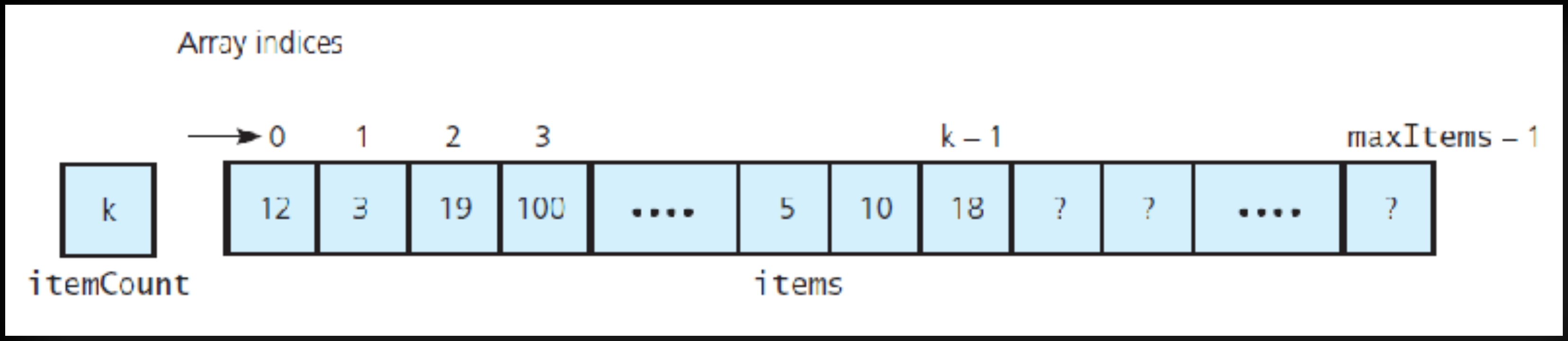




```
bag_contents.push_back(items_[0])

bag_contents.push_back(items_[1])

bag_contents.push_back(items_[2])

        . . .
```

Implementation

```
#include “ArrayBag.h”


// private
template<class ItemType>
int ArrayBag<ItemType>::getIndexOf(const ItemType& 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
```


Array indices
0
1
2
3
k – 1
maxItems – 1
k
12
3
19
100
....
5
10
18
?
?
....
?
itemCount
items