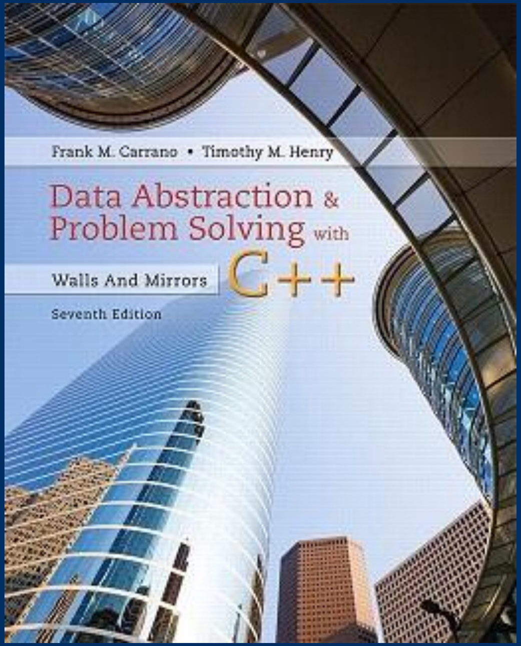
Chapter 3

Array-Based Implementations

CS 302 - Data Structures

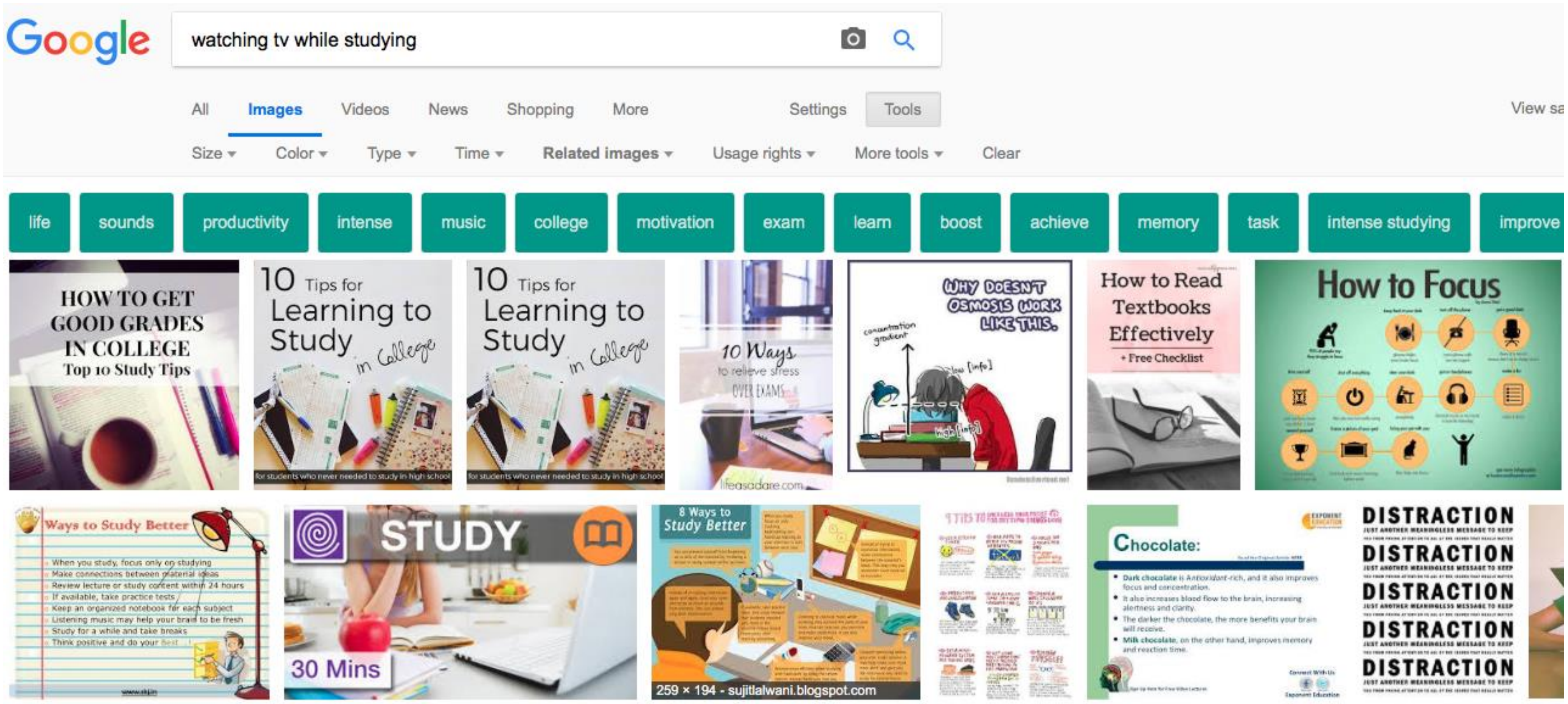
M. Abdullah Canbaz


Frank M. Carrano • Timothy M. Henry
Data Abstraction & Problem Solving with C++
Walls And Mirrors
Seventh Edition

Modified from authors' slides

Reminders

- Assignment 2 is available
  - Due Feb 14th at 2pm
- TA
  - Shehryar Khattak,
    Email: *shehryar [at] nevada {dot} unr {dot} edu,*
    Office Hours: Friday, 11:00 am - 1:00 pm at ARF 116
- Quiz 2 on Wednesday

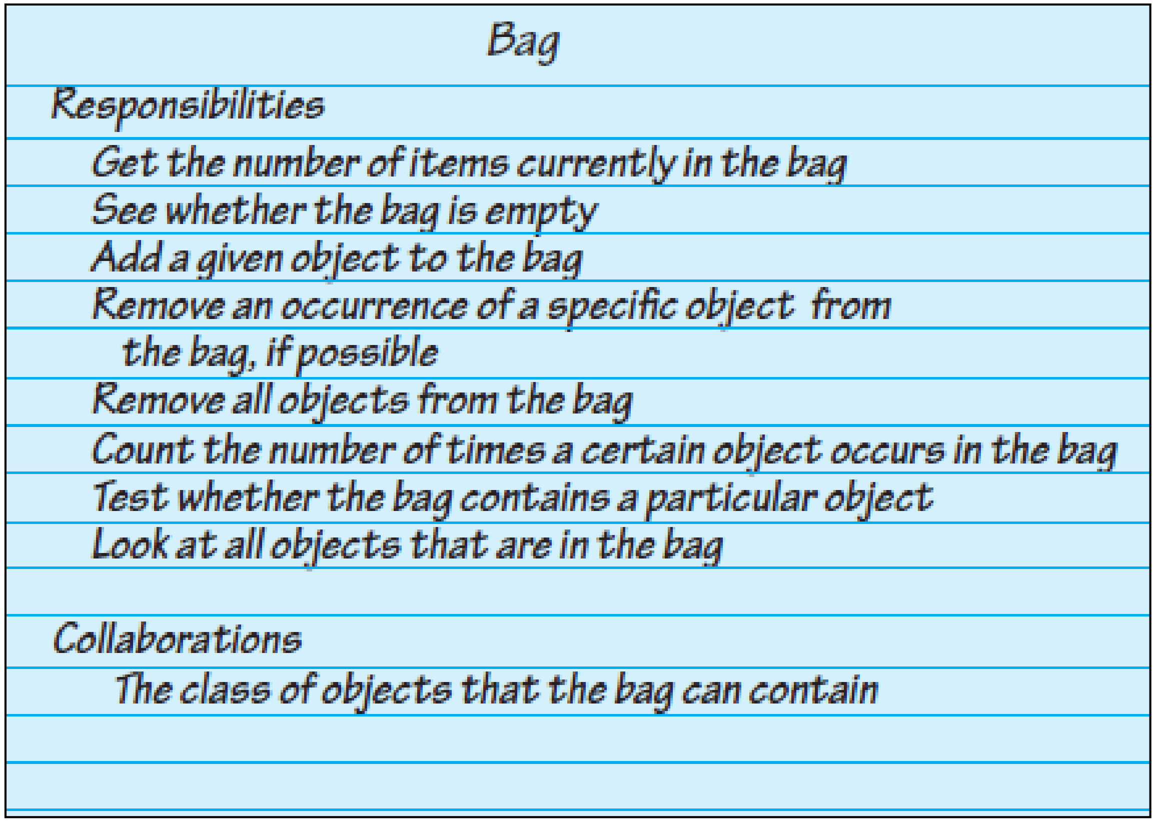

Google
watching tv while studying
All
Images
Videos
News
Shopping
More
Settings
Tools
View sa
Size
Color
Type
Time
Related images
Usage rights
More tools
Clear
life
sounds
productivity
intense
music
college
motivation
exam
learn
boost
achieve
memory
task
intense studying
improve
HOW TO GET GOOD GRADES IN COLLEGE Top 10 Study Tips
10 Tips for Learning to Study in College
10 Tips for Learning to Study in College
10 Ways to relieve stress OVER EXAMS
WHY DOESN'T OSMOSIS WORK LIKE THIS.
How to Read Textbooks Effectively
+ Free Checklist
How to Focus
Ways to Study Better
STUDY
30 Mins
8 Ways to Study Better
259 × 194 - sujitlalwani.blogspot.com
Chocolate:
DISTRACTION
DISTRACTION
DISTRACTION
DISTRACTION
DISTRACTION

The Approach

- An ADT is
  - A collection of data … and …
  - A set of operations on that data
- Specifications indicate
  - What ADT operations do
  - But not how to implement
- First step for implementation
  - Choose data structure

Recall - Identifying Behaviors

- A CRC card for a class **`Bag`**



Bag

Responsibilities

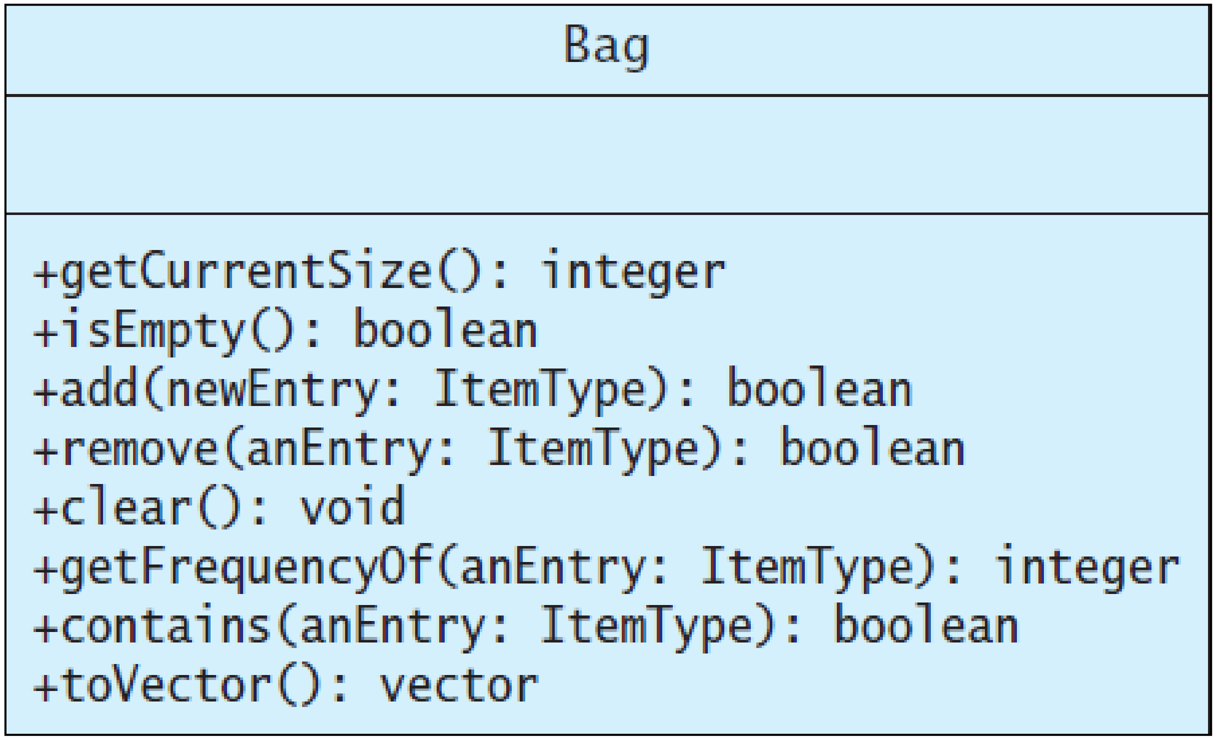
Get the number of items currently in the bag
See whether the bag is empty
Add a given object to the bag
Remove an occurrence of a specific object from the bag, if possible
Remove all objects from the bag
Count the number of times a certain object occurs in the bag
Test whether the bag contains a particular object
Look at all objects that are in the bag

Collaborations

The class of objects that the bag can contain

Recall - Specifying Data and Operations

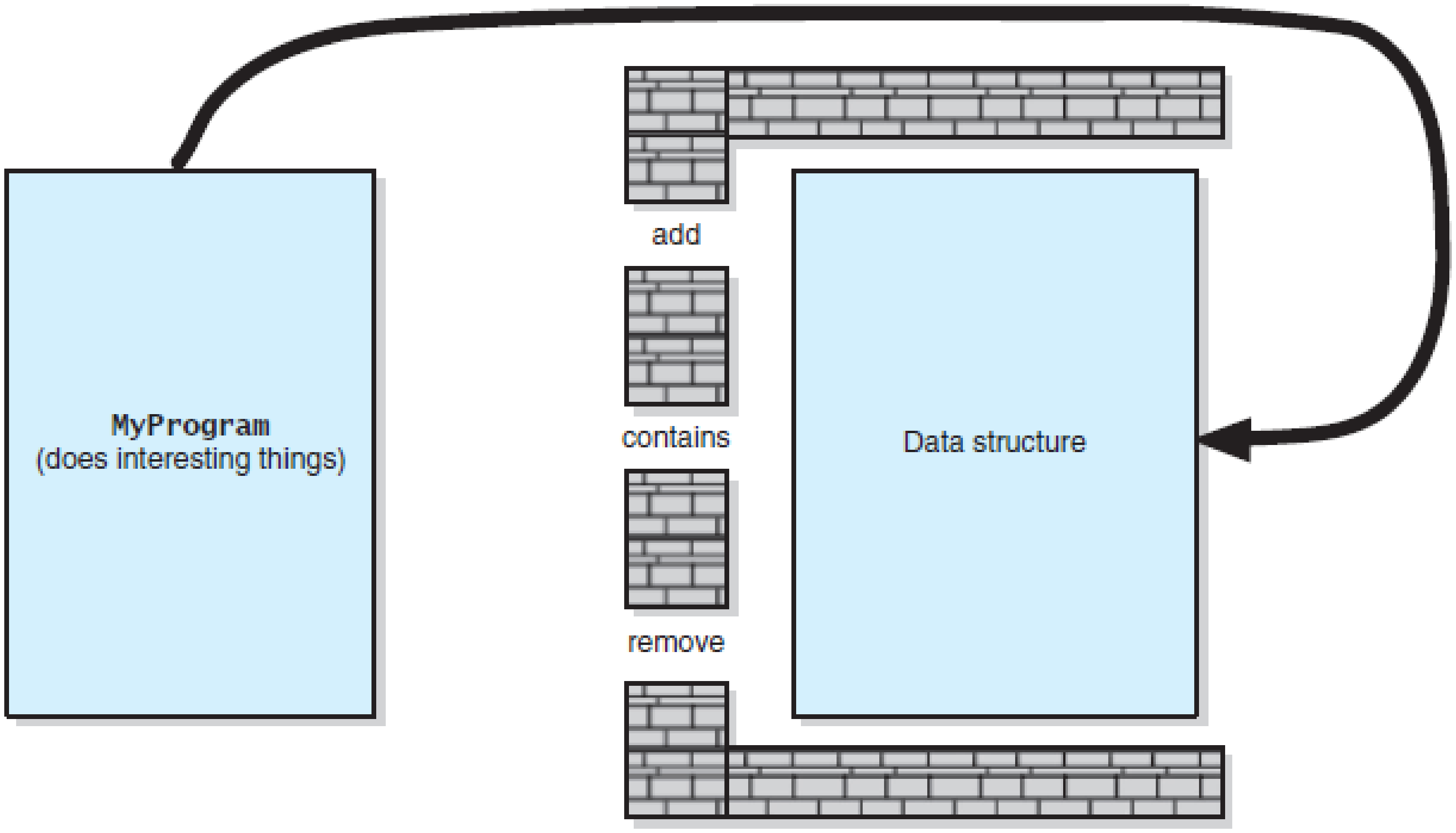
- UML notation for the class **`Bag`**



```
Bag

+getCurrentSize(): integer
+isEmpty(): boolean
+add(newEntry: ItemType): boolean
+remove(anEntry: ItemType): boolean
+clear(): void
+getFrequencyOf(anEntry: ItemType): integer
+contains(anEntry: ItemType): boolean
+toVector(): vector
```

The Approach

- Violating the wall of ADT operations



Core Methods

Poor approach

- Define entire class and attempt test

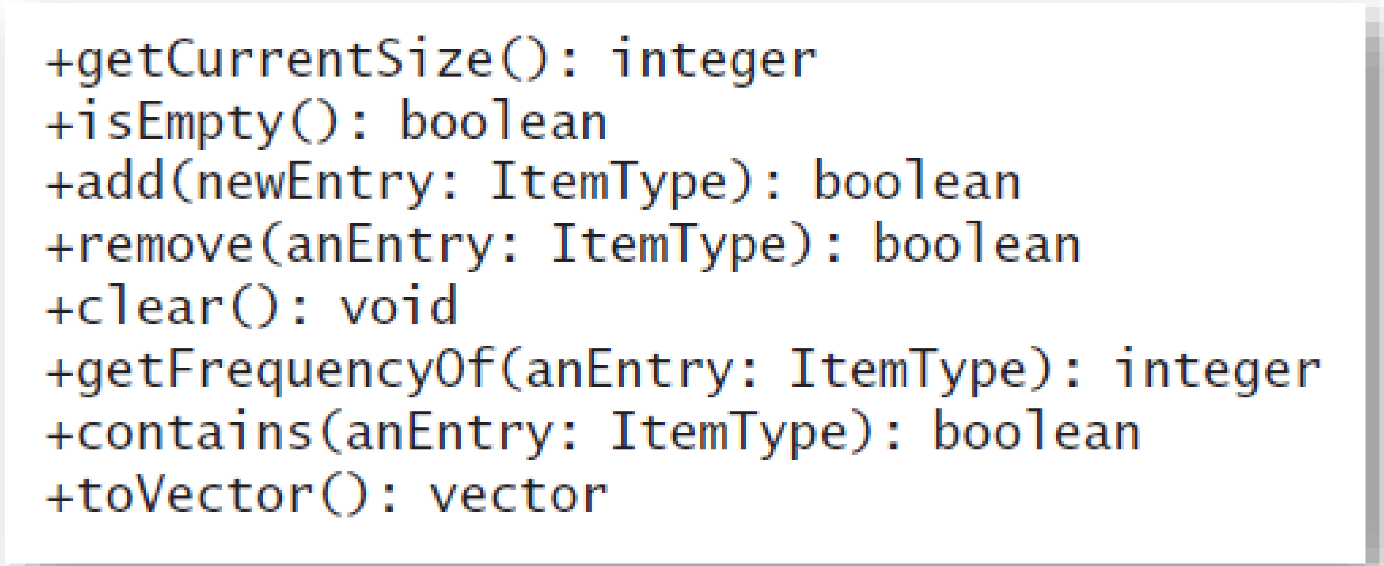
Better plan

- Identify, then test basic (core) methods
  - Create the container (constructors)
  - Add items
  - Display/list items
  - Remove items

Using Fixed-Size Arrays

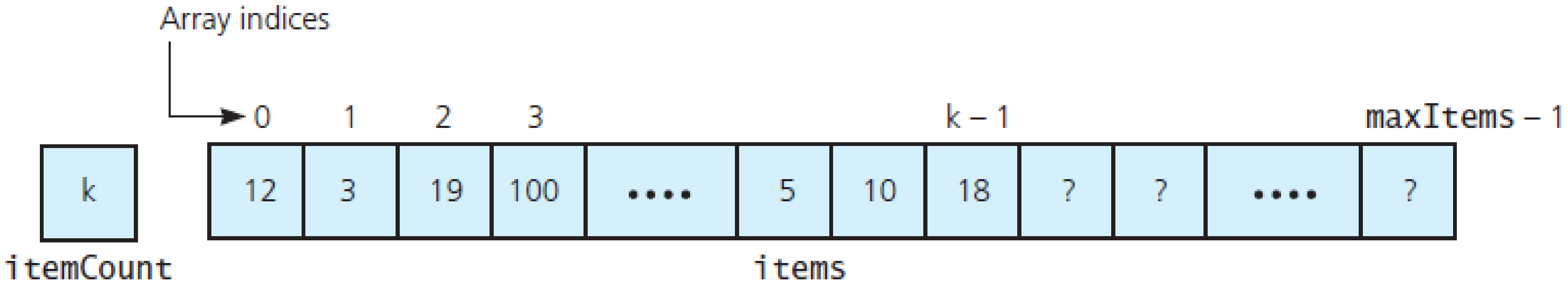
- Must keep track of array elements used, available
- Decide if first object goes in element 0 or 1
- Consider if the *add* method places elements in consecutive elements of array
- What happens when *add* method has used up final available element?

Array-Based Implementation

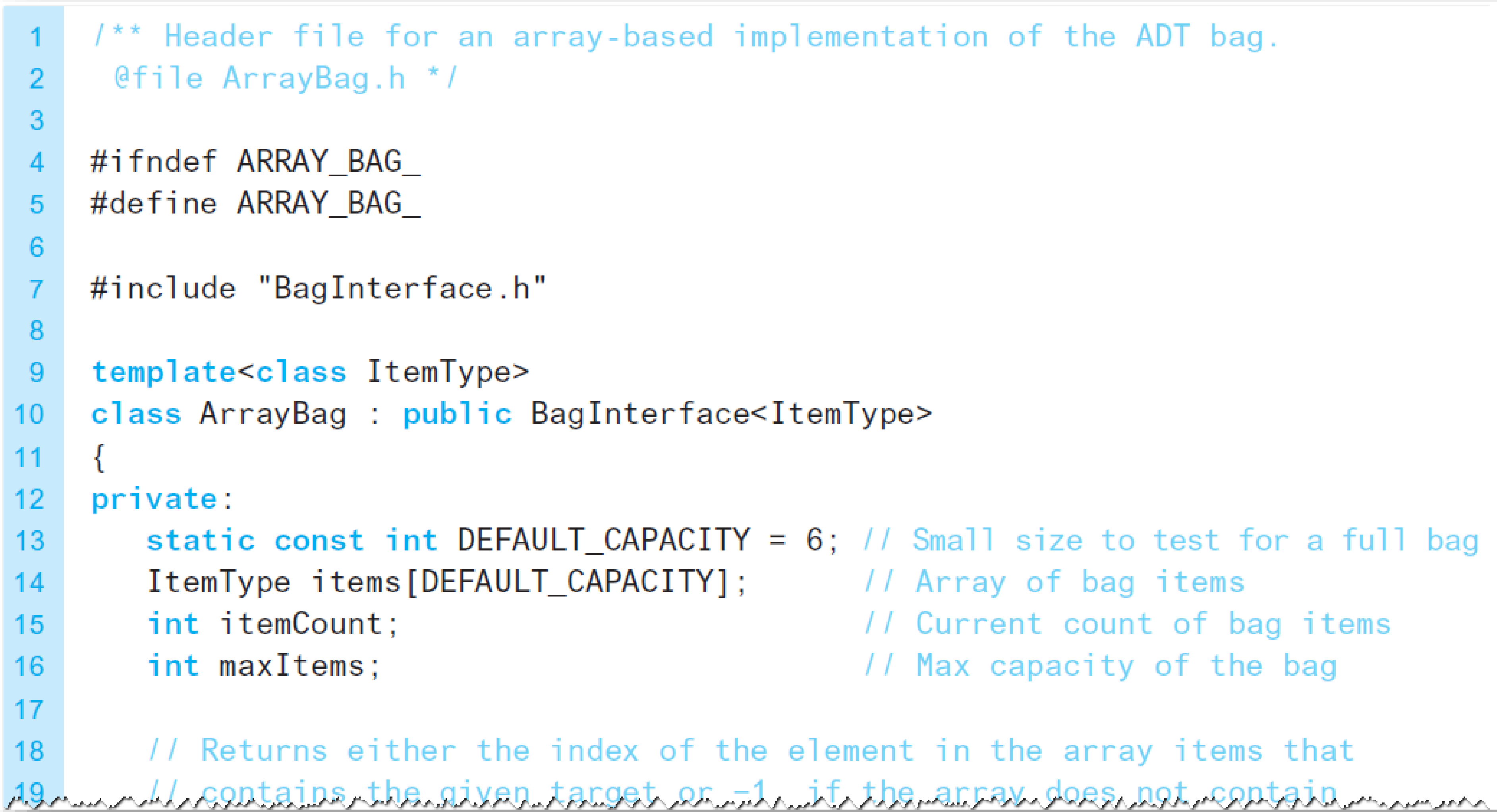


```
+getCurrentSize(): integer
+isEmpty(): boolean
+add(newEntry: ItemType): boolean
+remove(anEntry: ItemType): boolean
+clear(): void
+getFrequencyOf(anEntry: ItemType): integer
+contains(anEntry: ItemType): boolean
+toVector(): vector
```

An array-based implementation of the ADT bag



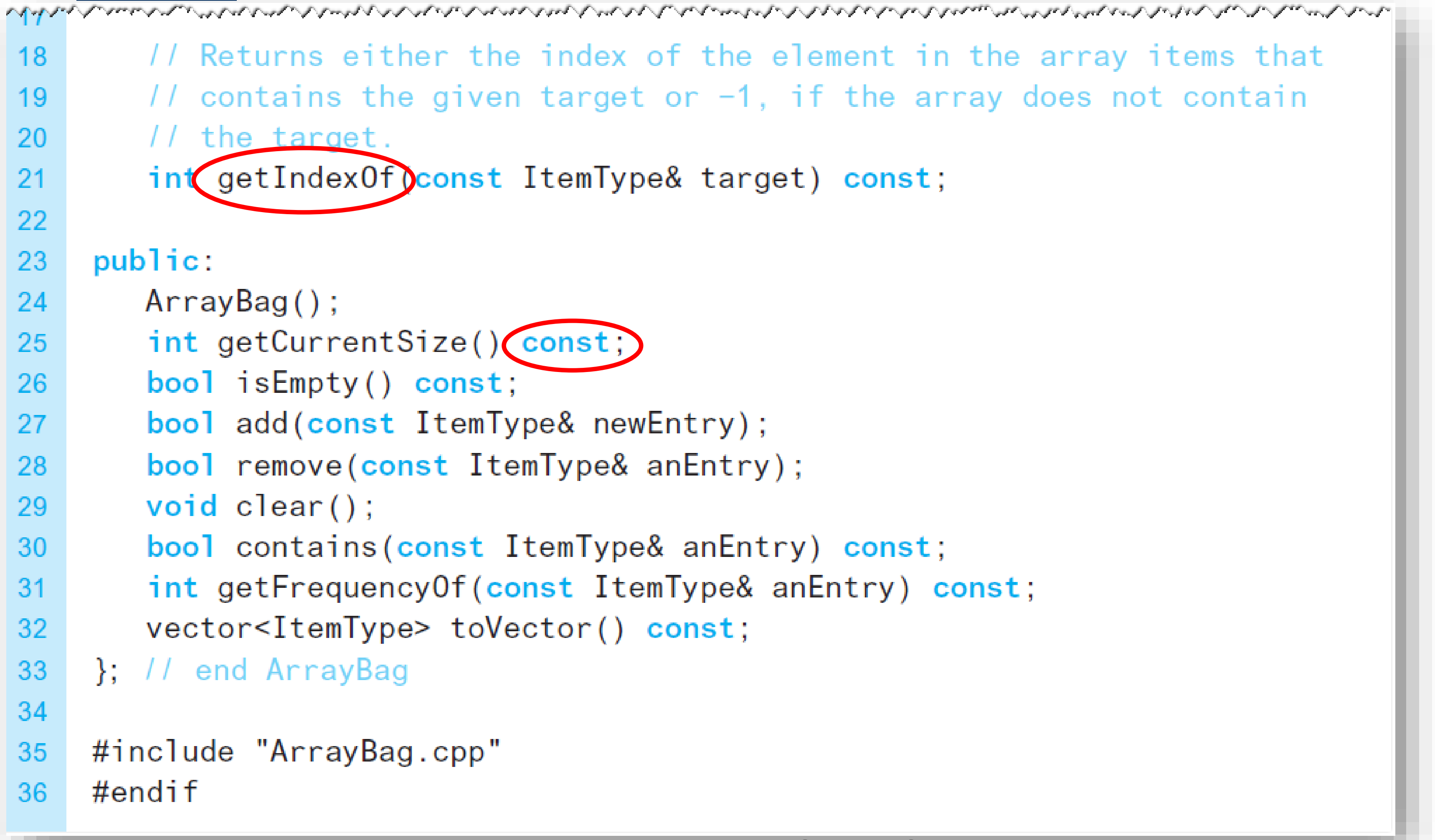
The Header File



```
/** Header file for an array-based implementation of the ADT bag.
 @file ArrayBag.h */

#ifndef ARRAY_BAG_
#define ARRAY_BAG_

#include "BagInterface.h"

template<class ItemType>
class ArrayBag : public BagInterface<ItemType>
{
private:
   static const int DEFAULT_CAPACITY = 6; // Small size to test for a full bag
   ItemType items[DEFAULT_CAPACITY];      // Array of bag items
   int itemCount;                         // Current count of bag items
   int maxItems;                          // Max capacity of the bag

   // Returns either the index of the element in the array items that
   // contains the given target or -1, if the array does not contain
```

LISTING 3-1 The header file for the class *ArrayBag*

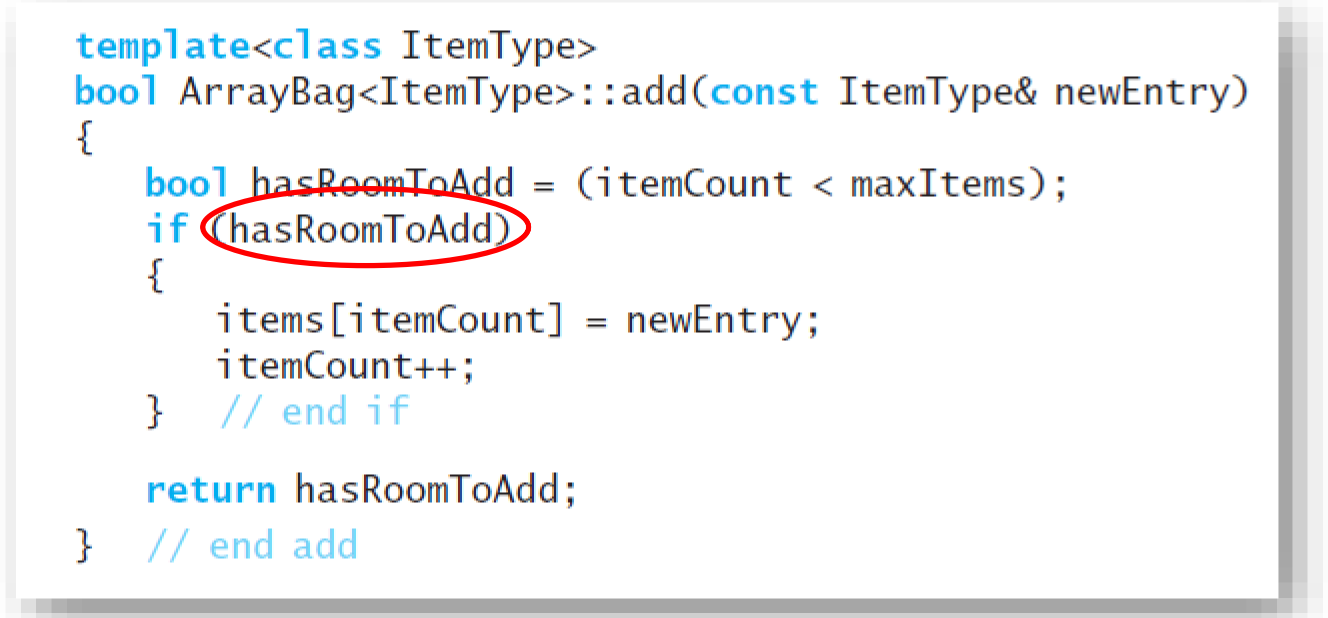
The Header File



```
   // Returns either the index of the element in the array items that
   // contains the given target or -1, if the array does not contain
   // the target.
   int getIndexOf(const ItemType& target) const;

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;
   vector<ItemType> toVector() const;
}; // end ArrayBag

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

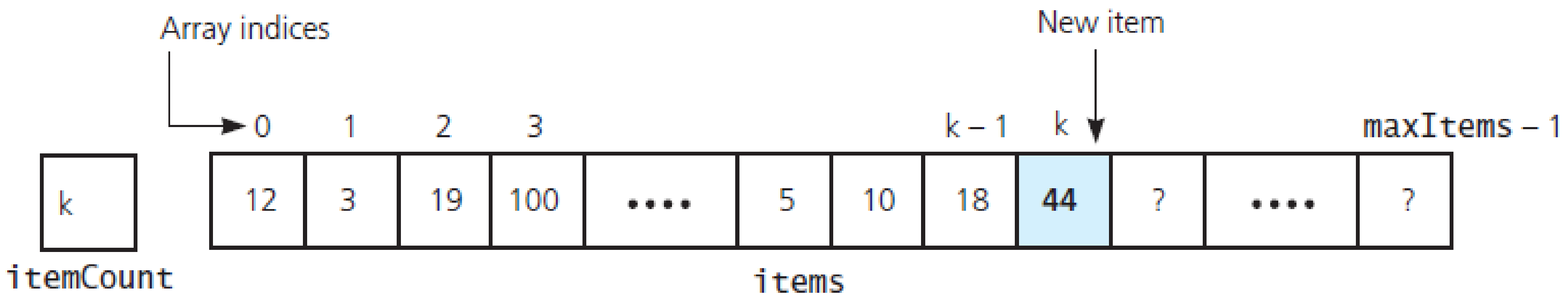
LISTING 3-1 The header file for the class *ArrayBag*

Defining the Core Methods

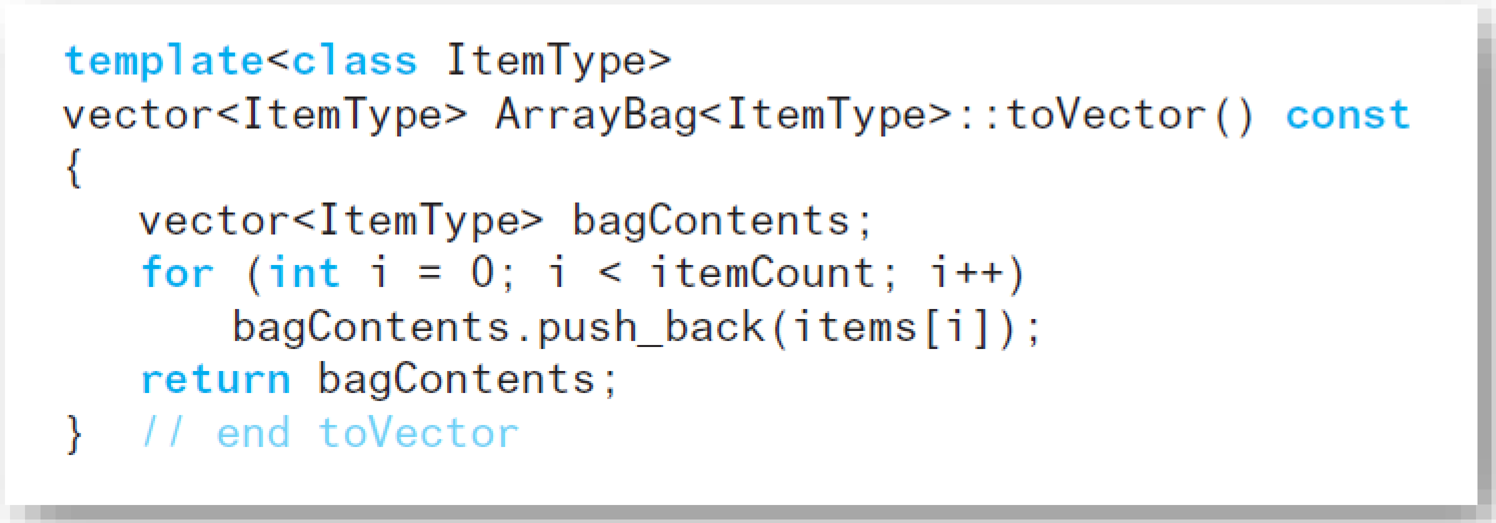


```
template<class ItemType>
bool ArrayBag<ItemType>::add(const ItemType& newEntry)
{
   bool hasRoomToAdd = (itemCount < maxItems);
   if (hasRoomToAdd)
   {
      items[itemCount] = newEntry;
      itemCount++;
   }  // end if

   return hasRoomToAdd;
}  // end add
```

Inserting a new entry into an array-based bag



Defining the Core Methods



```
template<class ItemType>
vector<ItemType> ArrayBag<ItemType>::toVector() const
{
   vector<ItemType> bagContents;
   for (int i = 0; i < itemCount; i++)
       bagContents.push_back(items[i]);
   return bagContents;
}  // end toVector
```

The method *toVector*

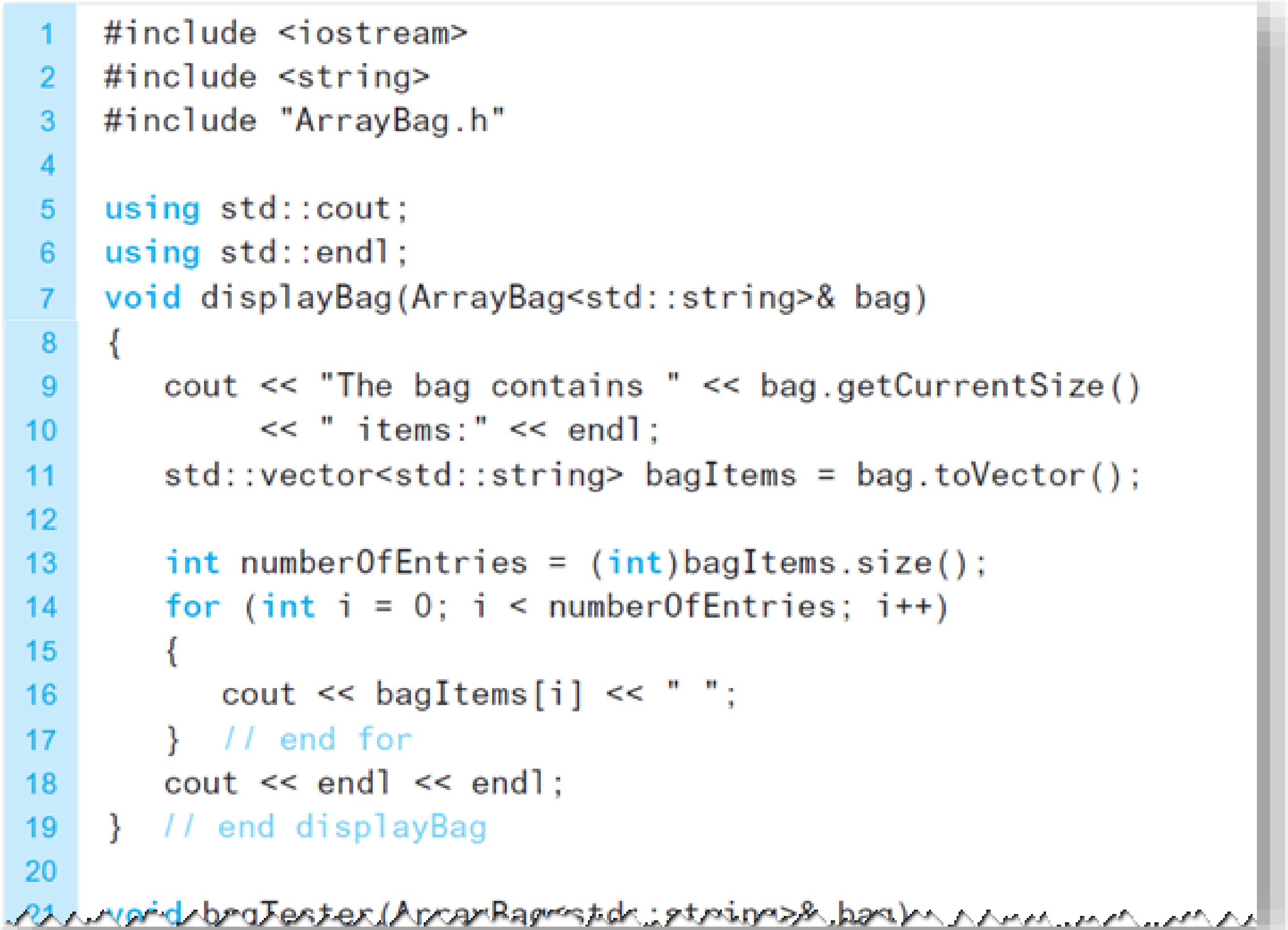
Defining the Core Methods

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

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

Methods *getCurrentSize* and *isEmpty*

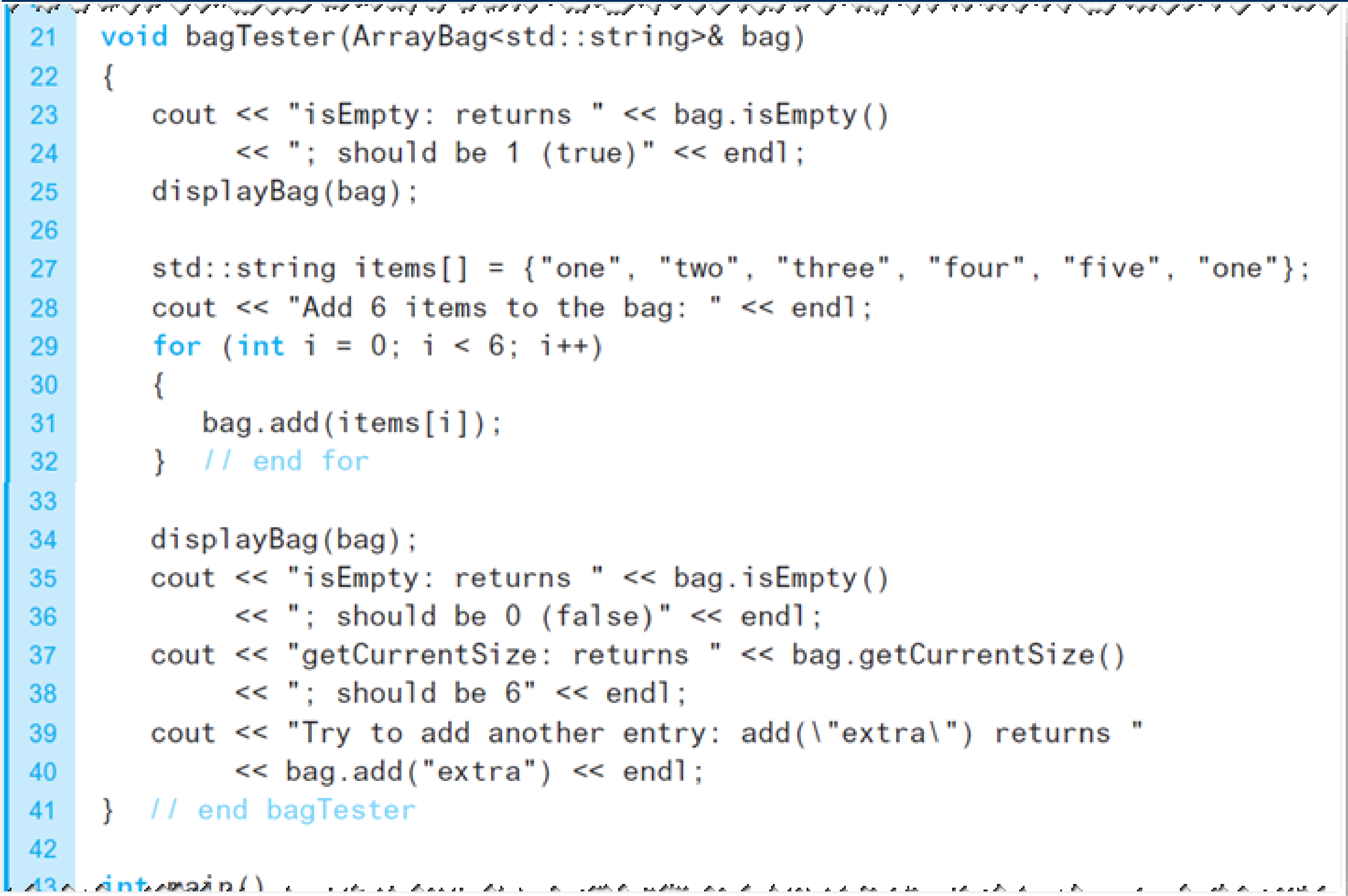
Testing the Core Methods



```
#include <iostream>
#include <string>
#include "ArrayBag.h"

using std::cout;
using std::endl;
void displayBag(ArrayBag<std::string>& bag)
{
   cout << "The bag contains " << bag.getCurrentSize()
        << " items:" << endl;
   std::vector<std::string> bagItems = bag.toVector();

   int numberOfEntries = (int)bagItems.size();
   for (int i = 0; i < numberOfEntries; i++)
   {
      cout << bagItems[i] << " ";
   }  // end for
   cout << endl << endl;
}  // end displayBag

void bagTester(ArrayBag<std::string>& bag)
```

LISTING 3-2 A program that tests the core methods of the class *ArrayBag*

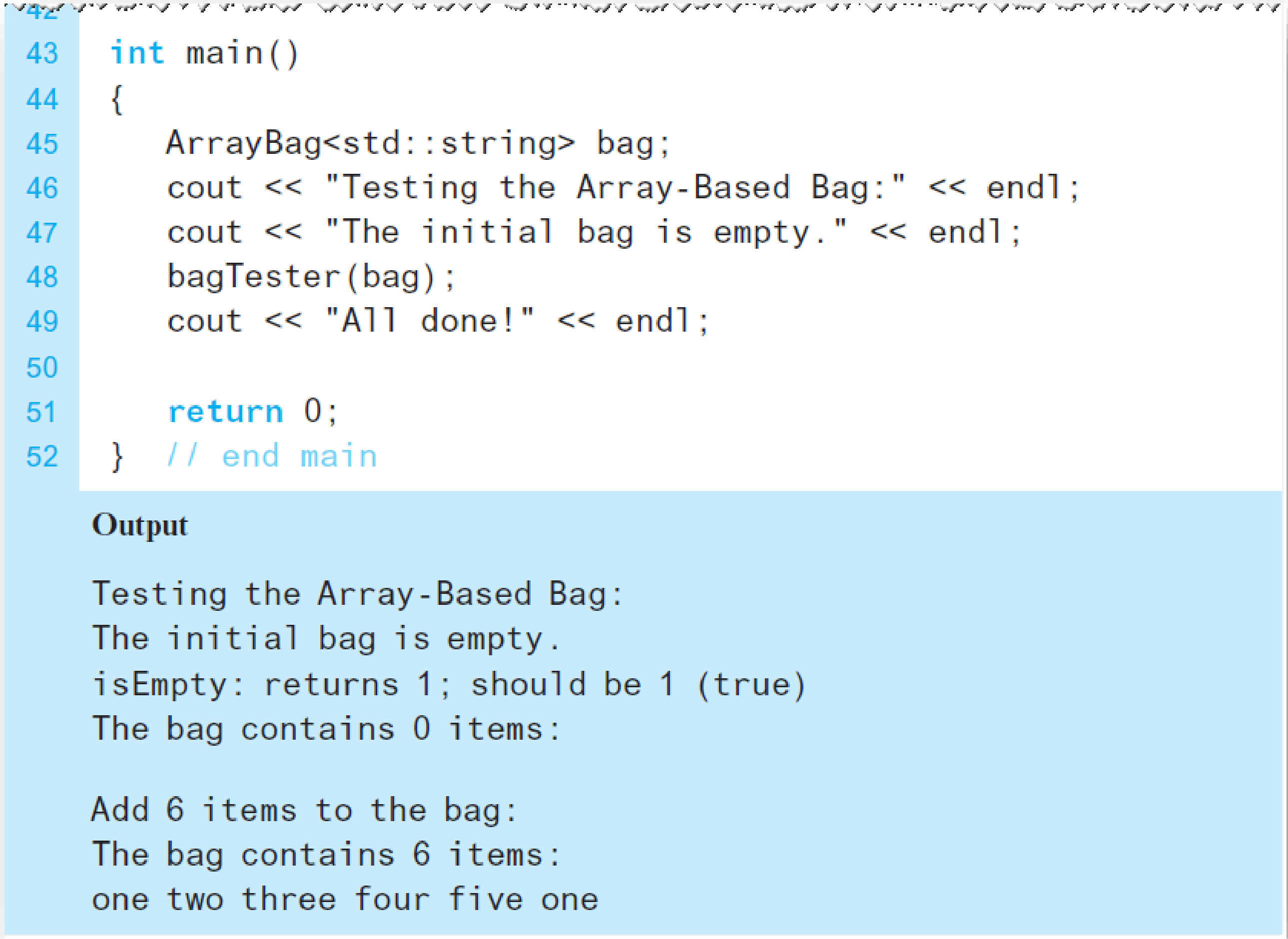
Testing the Core Methods



```
void bagTester(ArrayBag<std::string>& bag)
{
   cout << "isEmpty: returns " << bag.isEmpty()
        << "; should be 1 (true)" << endl;
   displayBag(bag);

   std::string items[] = {"one", "two", "three", "four", "five", "one"};
   cout << "Add 6 items to the bag: " << endl;
   for (int i = 0; i < 6; i++)
   {
      bag.add(items[i]);
   }  // end for

   displayBag(bag);
   cout << "isEmpty: returns " << bag.isEmpty()
        << "; should be 0 (false)" << endl;
   cout << "getCurrentSize: returns " << bag.getCurrentSize()
        << "; should be 6" << endl;
   cout << "Try to add another entry: add(\"extra\") returns "
        << bag.add("extra") << endl;
}  // end bagTester

int main()
```

LISTING 3-2 A program that tests the core methods of the class *ArrayBag*

Testing the Core Methods



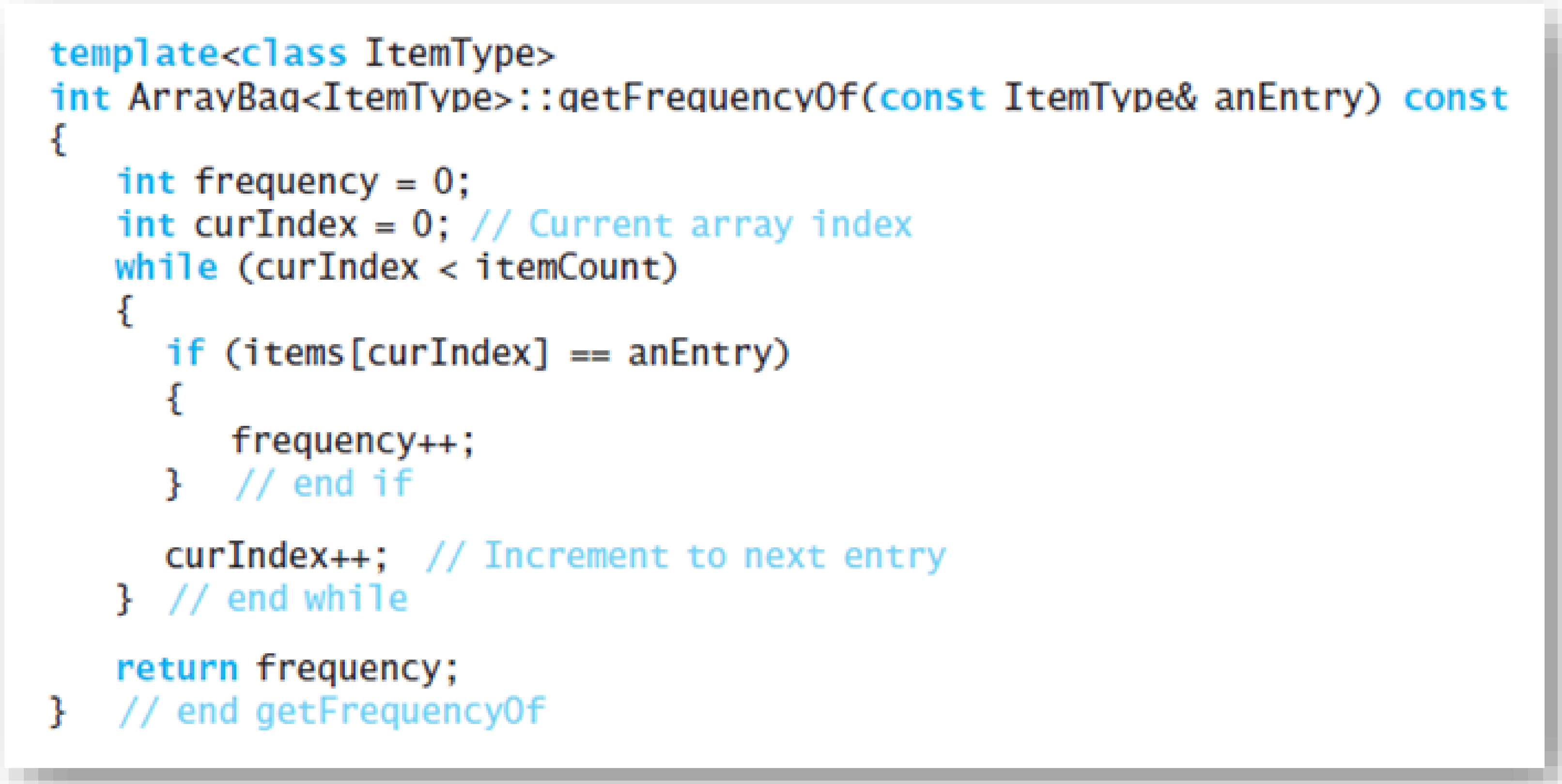
```cpp
int main()
{
   ArrayBag<std::string> bag;
   cout << "Testing the Array-Based Bag:" << endl;
   cout << "The initial bag is empty." << endl;
   bagTester(bag);
   cout << "All done!" << endl;

   return 0;
}  // end main
```

Output

```
Testing the Array-Based Bag:
The initial bag is empty.
isEmpty: returns 1; should be 1 (true)
The bag contains 0 items:

Add 6 items to the bag:
The bag contains 6 items:
one two three four five one
```

LISTING 3-2 A program that tests the core methods of the class *ArrayBag*

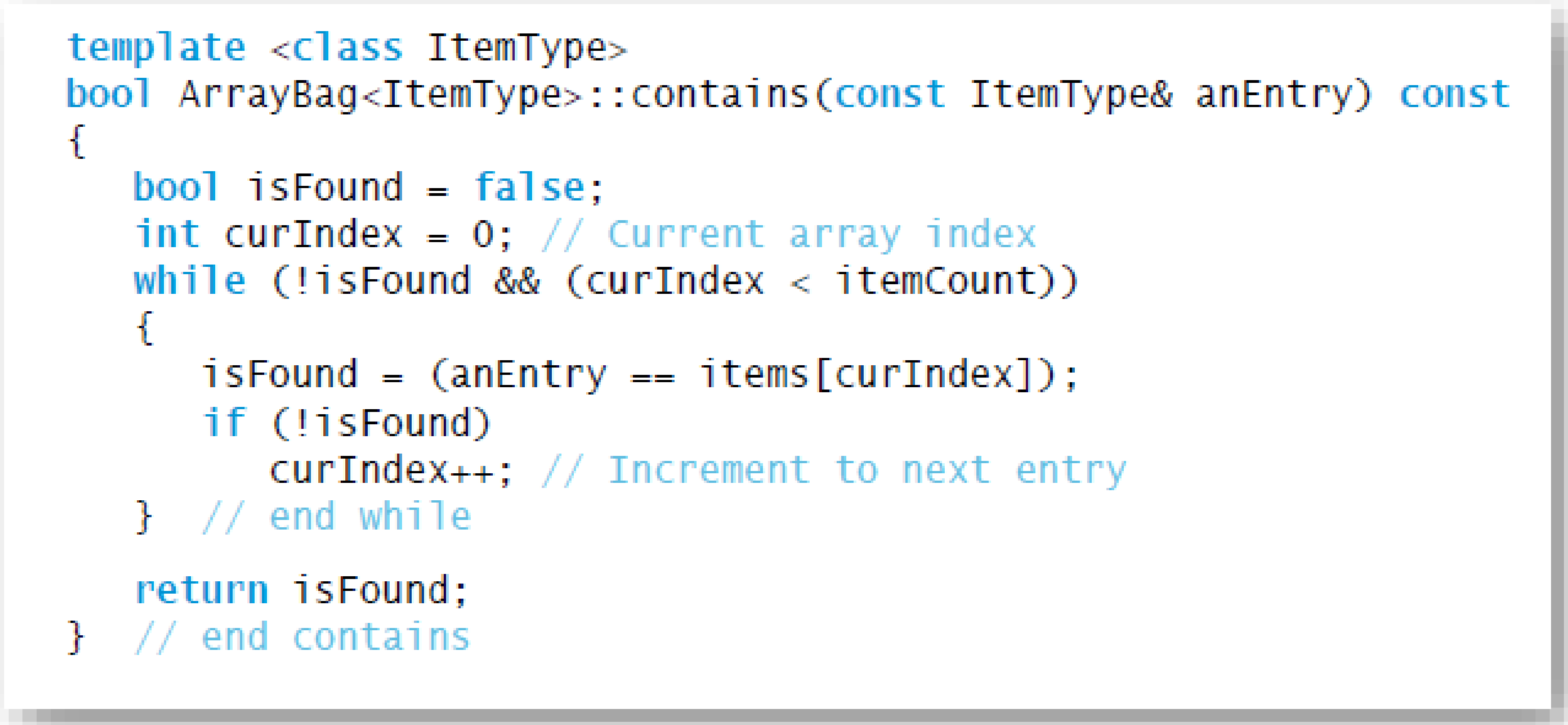
Implementing More Methods



```
template<class ItemType>
int ArrayBag<ItemType>::getFrequencyOf(const ItemType& anEntry) const
{
    int frequency = 0;
    int curIndex = 0; // Current array index
    while (curIndex < itemCount)
    {
       if (items[curIndex] == anEntry)
       {
          frequency++;
       }  // end if

       curIndex++;  // Increment to next entry
    }  // end while

    return frequency;
}   // end getFrequencyOf
```

Method *getFrequencyOf*

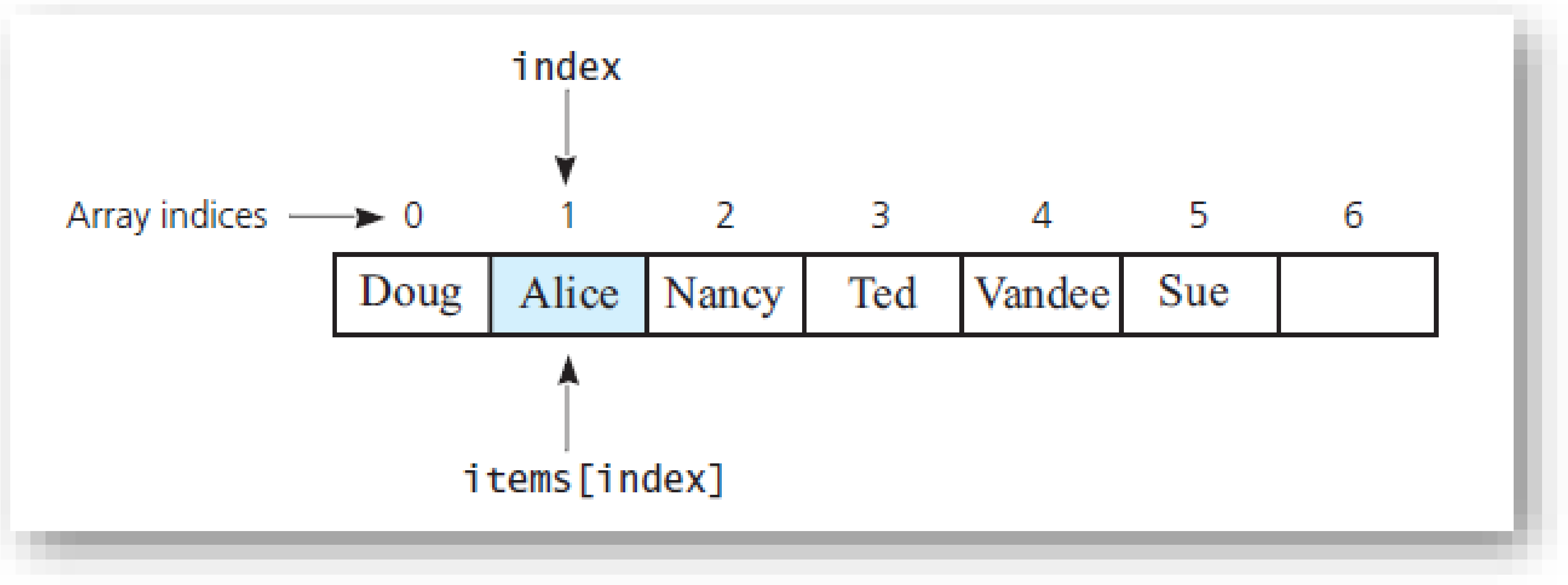
Implementing More Methods



```
template <class ItemType>
bool ArrayBag<ItemType>::contains(const ItemType& anEntry) const
{
   bool isFound = false;
   int curIndex = 0; // Current array index
   while (!isFound && (curIndex < itemCount))
   {
      isFound = (anEntry == items[curIndex]);
      if (!isFound)
         curIndex++; // Increment to next entry
   }  // end while

   return isFound;
}  // end contains
```

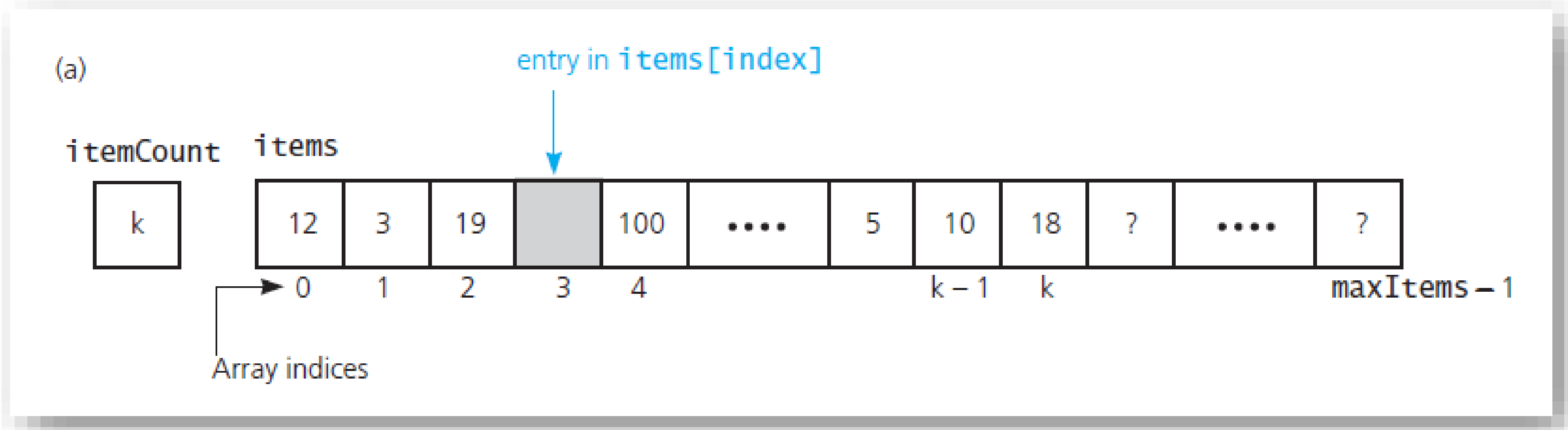
Possible implementation of method *contains*

Methods That Remove Entries



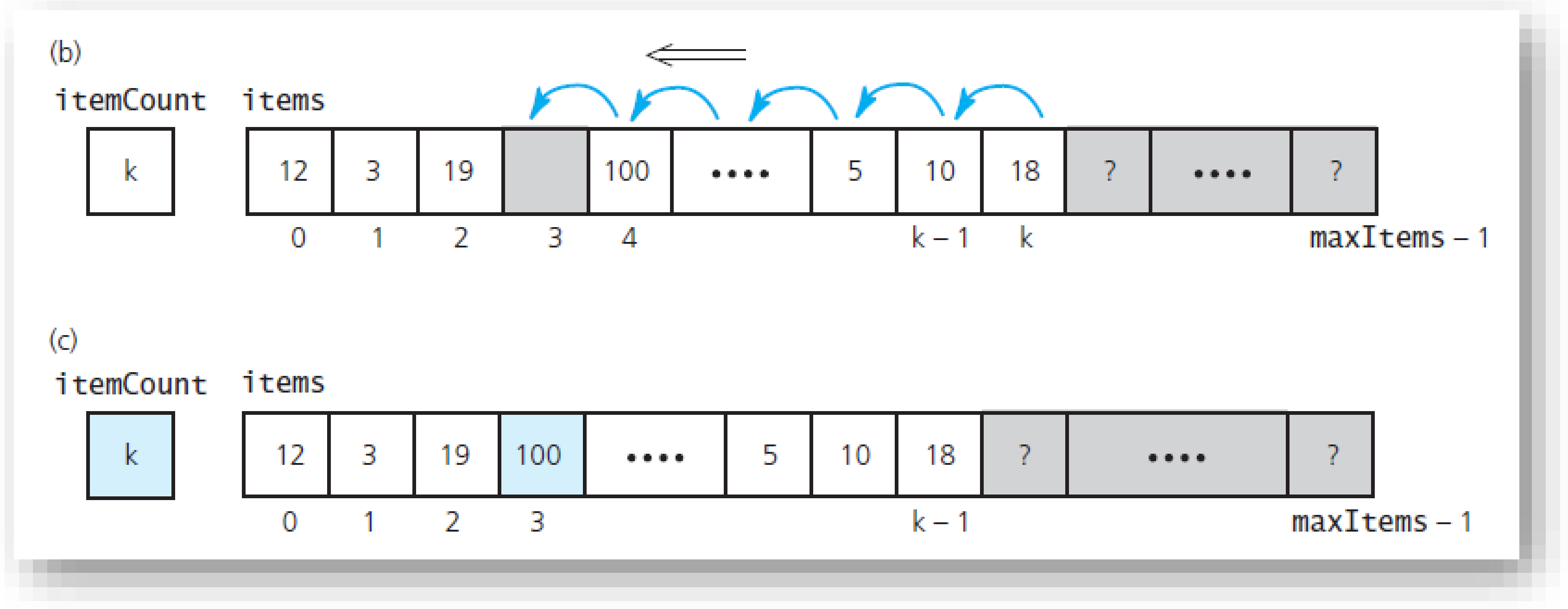
The array items after a successful search for the string "Alice"

Methods That Remove Entries



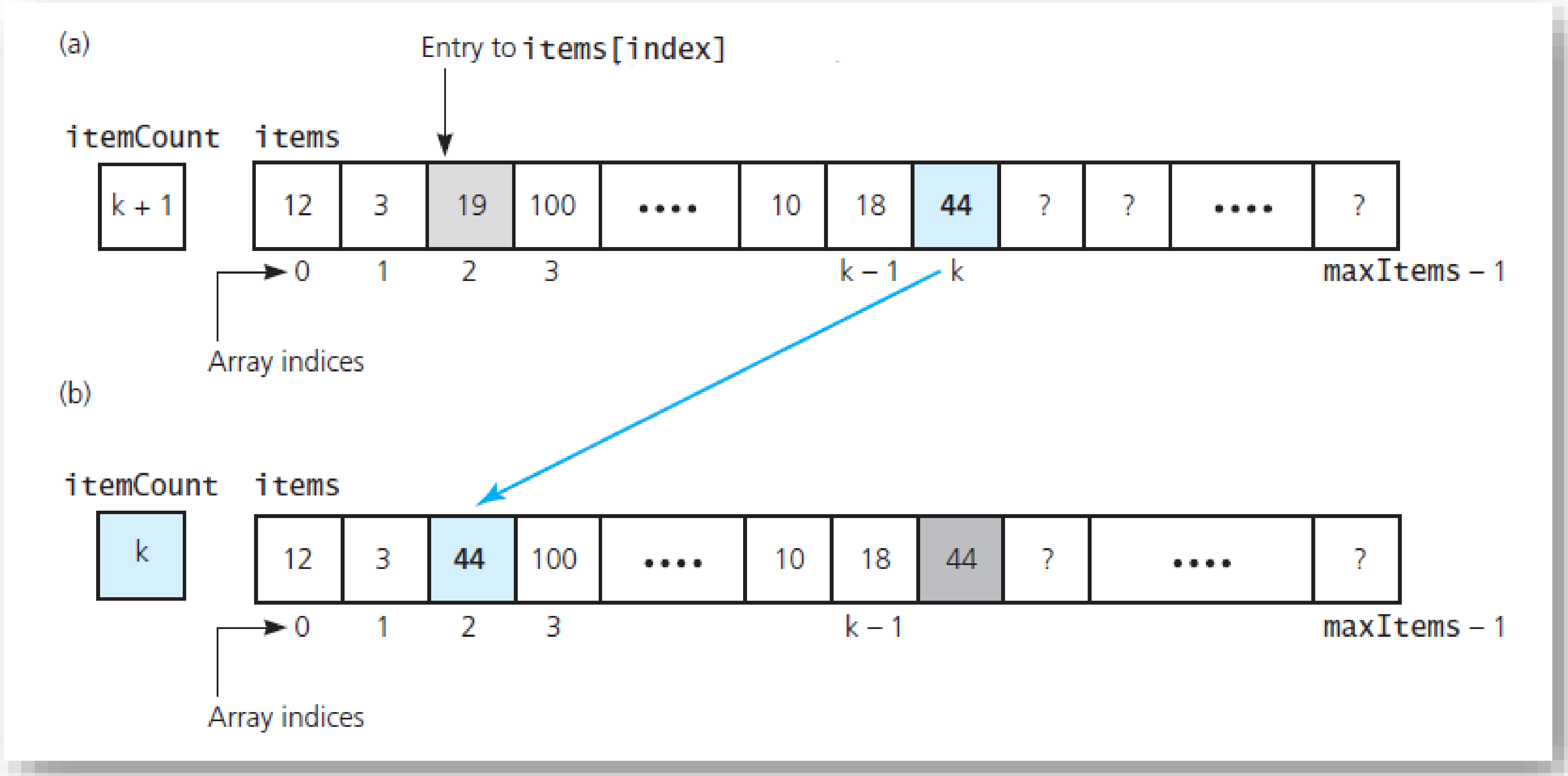
A gap in the array items after the entry in *items[index]* and decrementing *itemCount*;

Methods That Remove Entries



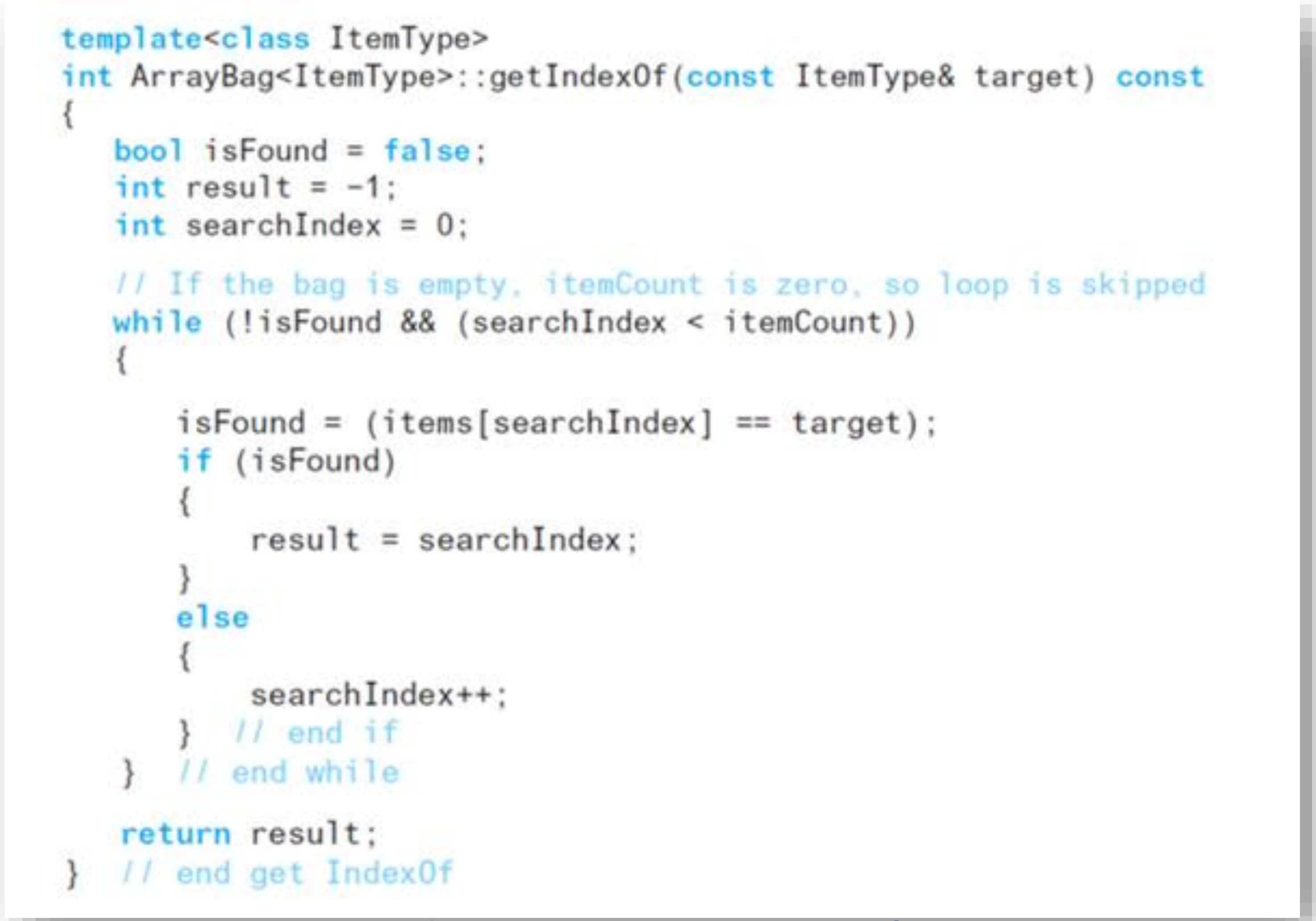
(b) shifting subsequent entries to avoid a gap;
(c) the array after shifting

Methods That Remove Entries



Avoiding a gap in the array while removing an entry

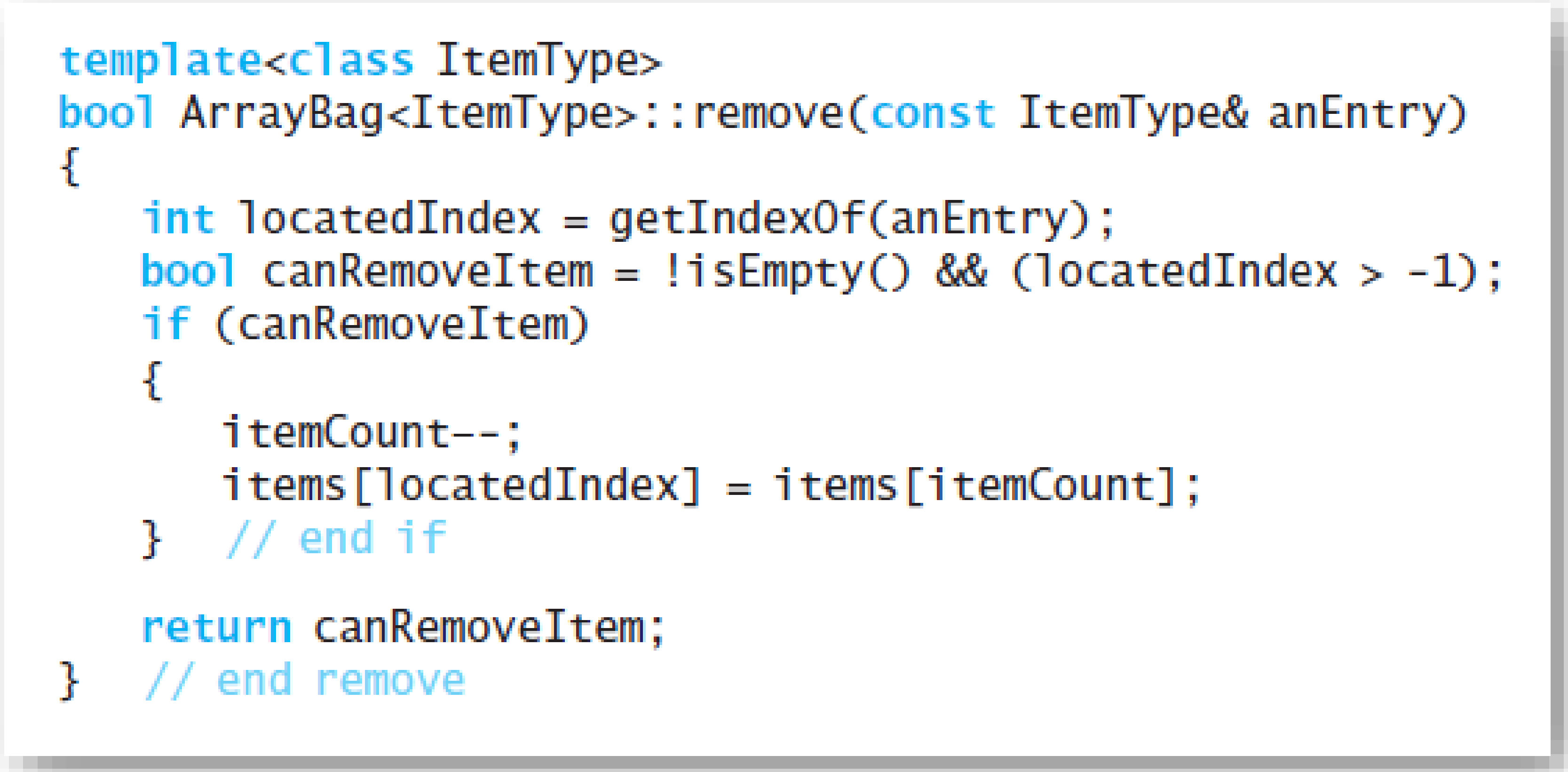
Methods That Remove Entries



```cpp
template<class ItemType>
int ArrayBag<ItemType>::getIndexOf(const ItemType& target) const
{
   bool isFound = false;
   int result = -1;
   int searchIndex = 0;

   // If the bag is empty, itemCount is zero, so loop is skipped
   while (!isFound && (searchIndex < itemCount))
   {

      isFound = (items[searchIndex] == target);
      if (isFound)
      {
         result = searchIndex;
      }
      else
      {
         searchIndex++;
      }  // end if
   }  // end while

   return result;
}  // end get IndexOf
```

Method getIndexOf

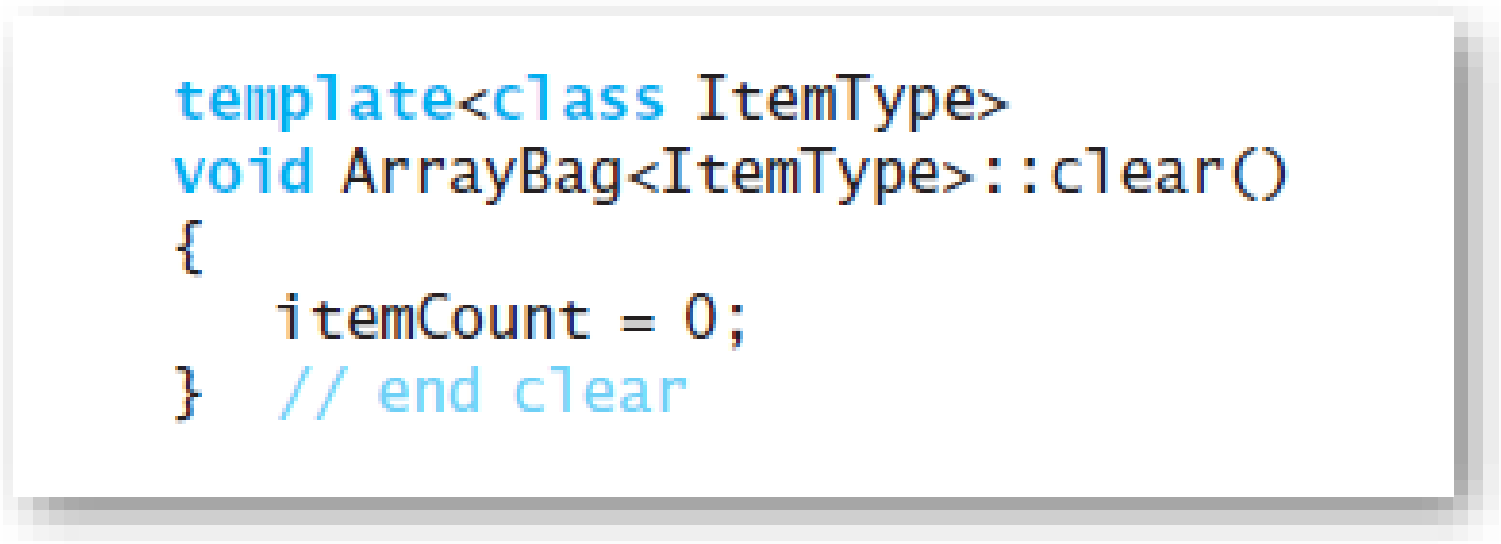
Methods That Remove Entries



```
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& anEntry)
{
    int locatedIndex = getIndexOf(anEntry);
    bool canRemoveItem = !isEmpty() && (locatedIndex > -1);
    if (canRemoveItem)
    {
        itemCount--;
        items[locatedIndex] = items[itemCount];
    }   // end if

    return canRemoveItem;
}   // end remove
```

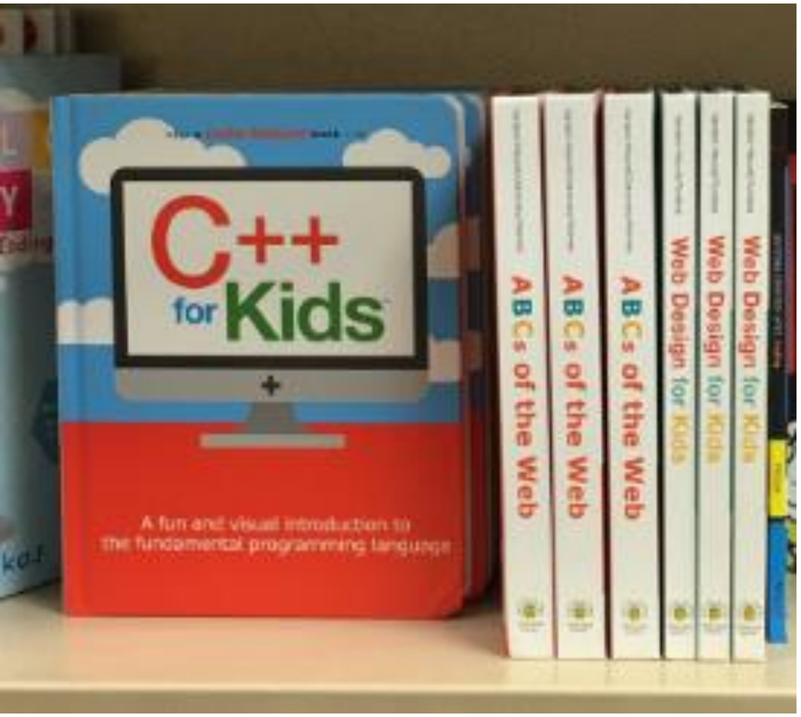
Method *remove*

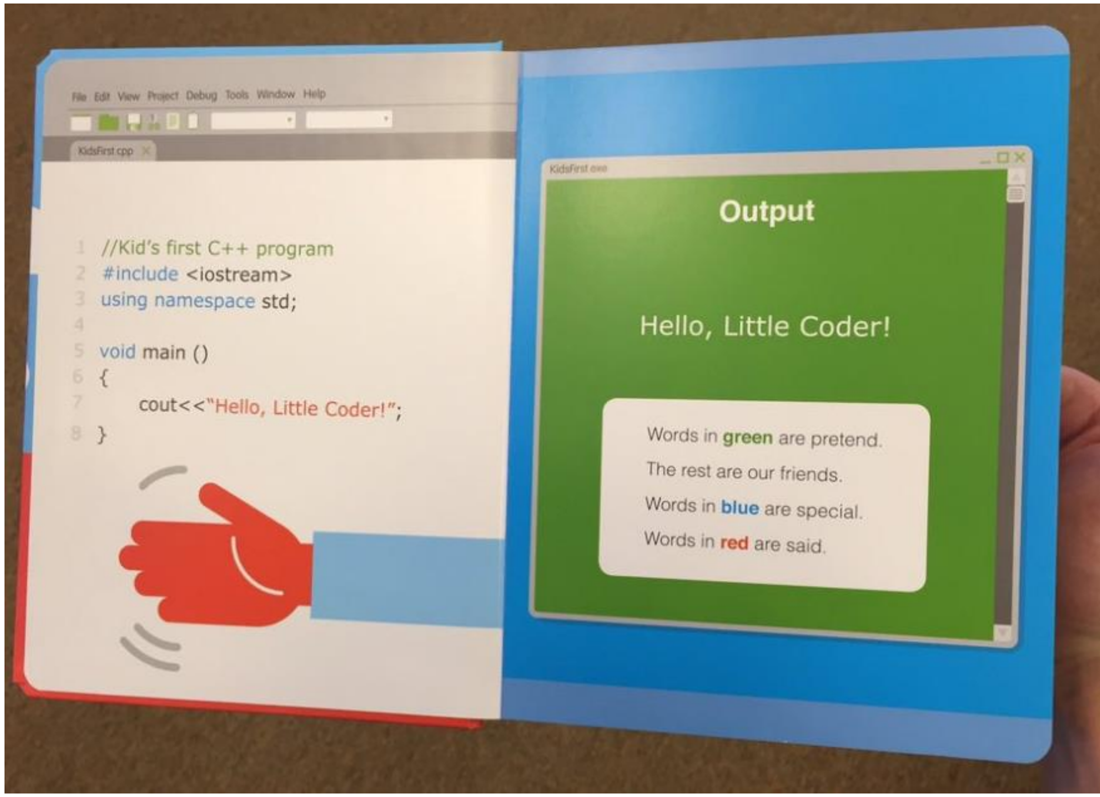
Methods That Remove Entries



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

Method *clear*




File Edit View Project Debug Tools Window Help
KidsFirst.cpp
KidsFirst.exe

```
//Kid's first C++ program
#include <iostream>
using namespace std;

void main ()
{
    cout<<"Hello, Little Coder!";
}
```

Output

```
Hello, Little Coder!
```

Words in **green** are pretend.
The rest are our friends.
Words in **blue** are special.
Words in **red** are said.