CSCI 2270
Data Structures and Algorithms
Lecture 7—many Bag functions

Elizabeth White
elizabeth.white@colorado.edu
Office hours: ECCS 128 or ECCS 112
Wed 9:30am-11:00am
Thurs 10:00am-11:30am

Administrivia

- HW1 posted
  - It will be due Sunday, Feb. 2nd
- HW2 will post this weekend; resizable Bag
  - It will be due Sunday, Feb. 9th
- Read pp.31-46 on C++ classes
- Did you get email from me at about 12:40 Monday?
  - Email me if not.
- Clicker scores have posted for Wednesday
  - Make sure to set the room frequency (AB)
  - Register your clicker

const functions

Notice that several of the member functions are marked const at the end:

```
bool ArrayBag<ItemType>::isEmpty() const
int ArrayBag<ItemType>::getCurrentSize() const
```

These functions are promising not to change the member variables of the Bag. Why can they do that?

We do this to make our code safer; accidental changes to the Bag get recognized and stopped at compile time.

const parameters

Notice that other member functions have const input parameters:

```
bool ArrayBag<ItemType>::add
        (const ItemType& newItem)
bool ArrayBag<ItemType>::remove
        (const ItemType& anItem)
```

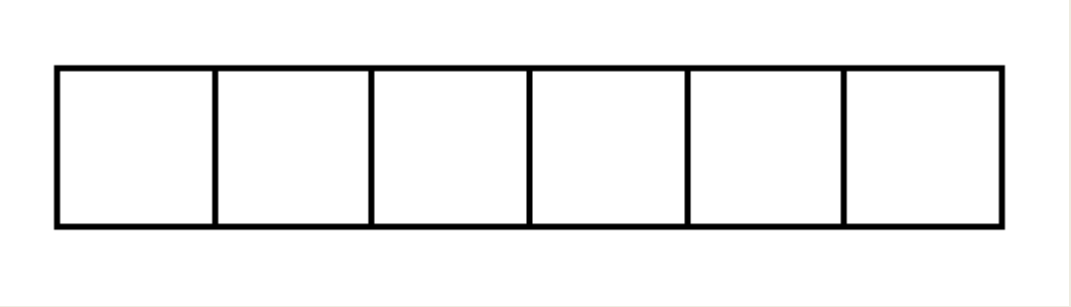
What are we promising here? The item we add or remove must not change. (The Bag changes, but the item just gets put into the items array.) Again, this protects against programmer errors.

Adding items

bool ArrayBag<ItemType>::add

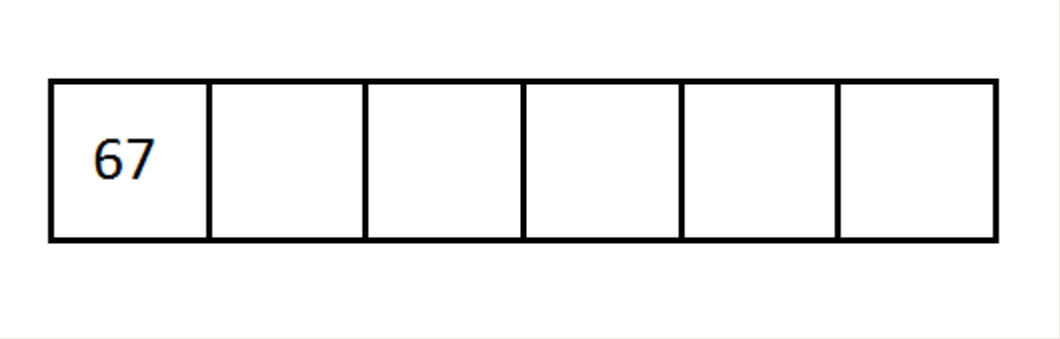
(const ItemType& newItem)

Empty Bag of ints, DEFAULT_CAPACITY=6, itemCount=0



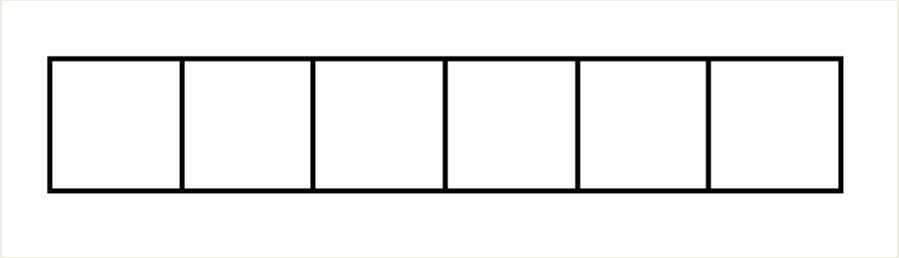
items

Add a 67; itemCount becomes 1. Gadzooks.



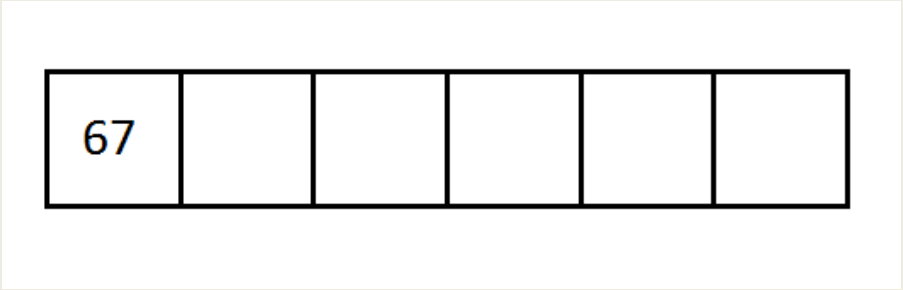
Adding items

Empty Bag of ints, DEFAULT_CAPACITY=6, itemCount=0



items

Add a 67; itemCount becomes 1



items

How do we get the 67 in the first position?

a) itemCount = 67;

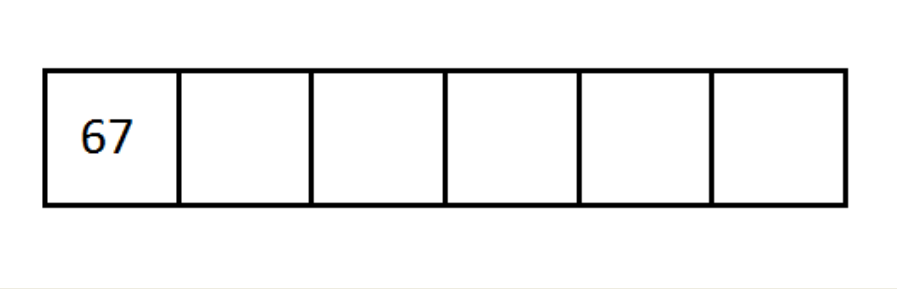
b) items[1] = 67;

c) items[0] = 67;

d) items[1] = itemCount;

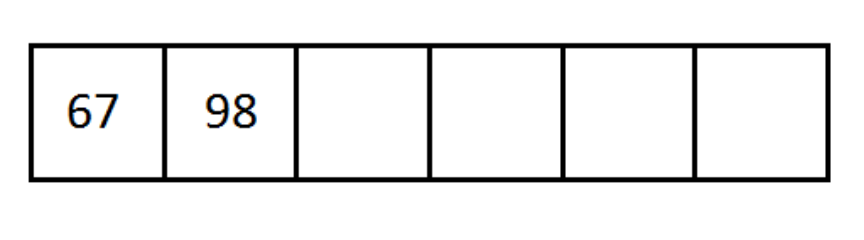
Adding items

Bag of ints, DEFAULT_CAPACITY=6, itemCount=1



items

Add a 98; itemCount becomes 2



items

How do we get the 98 in the second position?

a) itemCount = 98;

b) items[1] = 98;

c) items[0] = 98;

d) items[1] = itemCount;

Adding items

Can we generalize this to any available slot in the Bag?

```
items[ ? ] = newItem;
itemCount = itemCount + 1;
return true;
```

Can we do it all in one line? Yes…

```
items[  ??  ] = newItem;
return true;
```

What if the Bag is full?

Add nothing and return false

Segmentation faults?

Check those indexes! Count from 0

Finding items

How can we tell if the Bag contains an item?

Write a loop to go through the items…

Note: corrected unsigned int to int

a)
```
for (~~unsigned~~ int q = 1; q <= itemCount; q++)
        if (items[q] == anItem) return true;
```

b)
```
for (~~unsigned~~ int q = 0; q <= itemCount; q++)
        if (items[q] == anItem) return true;
```

c)
```
for (~~unsigned~~ int q = 1; q < itemCount; q++)
        if (items[q] == anItem) return true;
```

d)
```
for (~~unsigned~~ int q = 0; q < itemCount; q++)
        if (items[q] == anItem) return true;
```

Finding items

How can we tell if the Bag does not contain an item?

If we get all the way through the loop, what can we assume?

What should we return?

Counting items

How can we tell how many copies of an items the Bag contains?

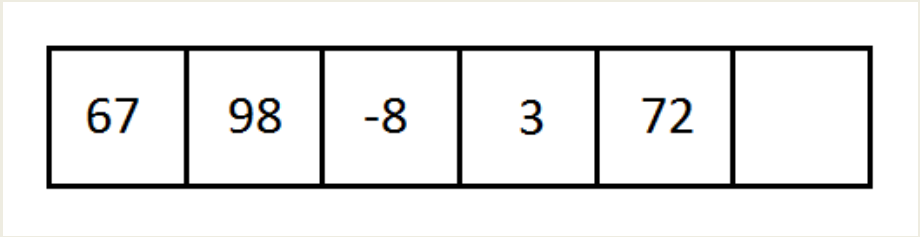
Similarity to contains: write a loop

Difference from contains: we have to check every slot

(why?)

Removing items

Bag of ints, DEFAULT_CAPACITY=6, itemCount=5

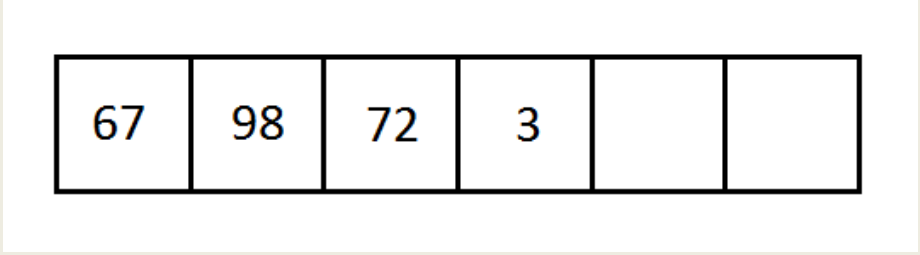


Suppose we want to remove the -8.

We write a loop to find the array position of the -8

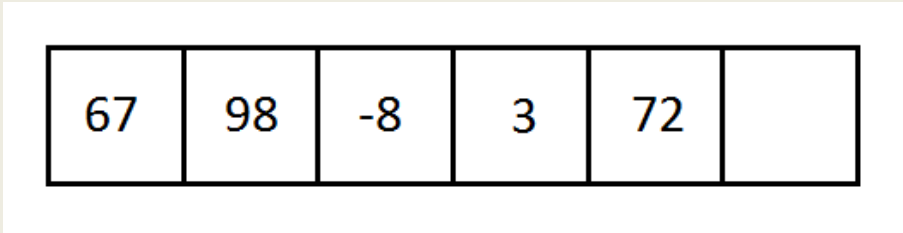
But removing the 8 makes a gap!

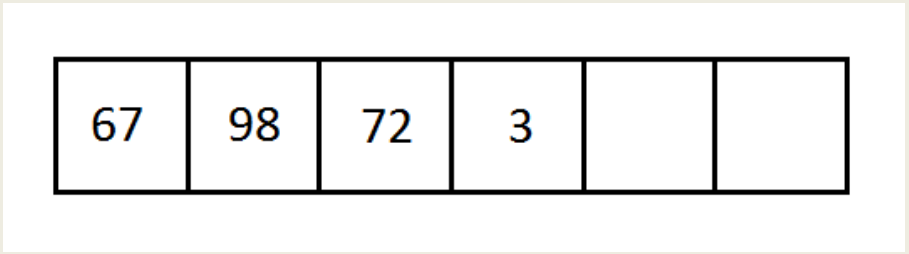
So we shift the last item over the -8 to fill the gap and decrease itemCount by 1.



Removing items

Suppose the item we're removing is at array position q, and the number of items we have is greater than q. What code would move the last item over the item we're removing?





a) items[0] = items[q];
b) items[q] = items[itemCount - 1];
c) items[q] = items[itemCount];
d) items[itemCount] = items[q];
e) items[itemCount - 1] = items[q];

Removing items

If the Bag were an ordered collection of items, what would be bad about this gap filling plan?

How would we preserve order while removing items in the Bag, if we had to? (Don't do this for hw1.)

How much work is preserving order, compared to what we did in the previous slides?

What would we have to watch out for here?

Listing items

Create an empty vector (another array-like template class in C++)

vector<ItemType> itemList;

Write a loop: for each item in the items array, add it to the itemList vector using a line like this (with no ?)

itemList.push_back(items[?]);

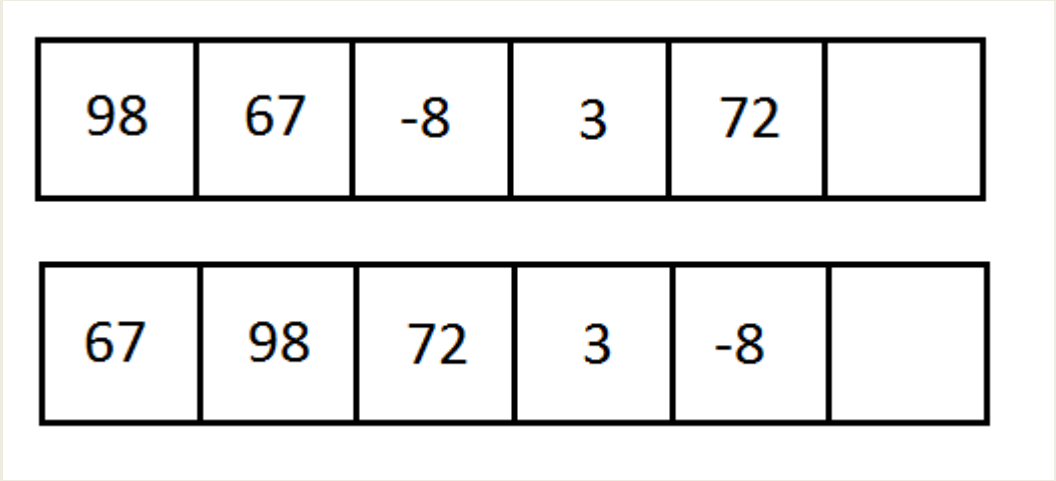
Seems a little redundant, no?

We're using an array to store data anyway

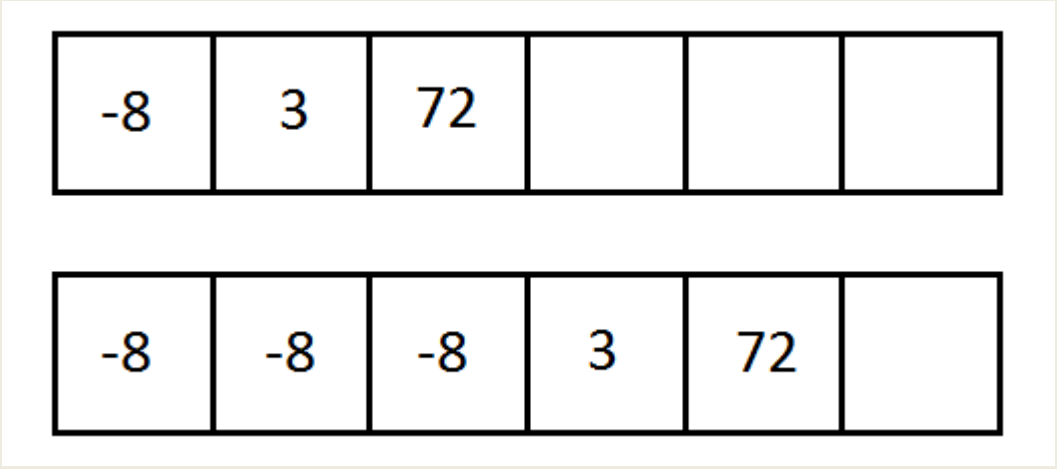
Later, when you write a B-tree container class, this routine will let you make a list from its items even if they're not stored in a linear way; then it will seem more useful.

Food for thought…

How would you decide if 2 Bags were equal?



equal Bags



not equal Bags