Container Classes

COMP2012

Lab 5

Container Classes

- A container class is a data type that is capable of holding a collection of items
- In C++, container classes can be implemented as a class, along with member functions to add, remove, and examine items

Bags

- For the first example, think about a bag
- Inside the bag are some numbers

- When you first begin to use a bag, the bag will be empty
- We count on this to be the **initial state** of any bag that we use

Inserting Numbers into a Bag

- Numbers may be inserted into a bag
- The bag can hold many numbers
- We can even insert the same number more than once

I am putting the number 4 into the bag
4
4
8
The 4 is in the bag
4
4
8

Examining a Bag

- We may ask about the contents of the bag.

Have you got any 4's?
4
8
4
Yes, I have two of them

Removing a Number from a Bag

- We may remove a number from a bag
- But we remove only one number at a time

This 4 is out of here
One 4 is gone but the other 4 remains
4
4
8

How Many Numbers

- Another operation is to determine how many numbers are in a bag

How many numbers do you have?
I have three numbers now
4
4
8

Summary of the Bag Operations

- A bag can be put in its **initial state**, which is an empty bag.
- Numbers can be **inserted** into the bag.
- You may check how many **occurrences** of a certain number are in the bag.
- Numbers can be **removed** from the bag.
- You can check **how many** numbers are in the bag.

The Bag Class

- C++ classes can be used to implement a container class such as a bag
- The class definition includes:
  - The heading of the definition
  - A constructor prototype
  - Prototypes for public member functions
  - Private member variables

```
class bag
{
public:
           bag(  );
           void insert(...);
           void remove(...);
           ...and so on
private:
           int data[CAPACITY];
           int count;
};
```

The Bag's Default Constructor

- Places a bag in the initial state (an empty bag)

```
bag::bag( )
// Postcondition: The bag has been initialized
// and it is now empty.
{
       . . .
}
```

The Insert and Remove Function

- Inserts a new number in the bag

```
void bag::insert(int new_entry)
// Precondition: The bag is not full.
// Postcondition: A new copy of new_entry has
// been added to the bag.
{
    . . .
}
```

- Removes one copy of a number

```
void bag::remove(int target)
// Postcondition: If target was in the bag, then
// one copy of target has been removed from the
// bag; otherwise the bag is unchanged.
{
    . . .
}
```

The Size and Occurrences Function

- Counts how many integers are in the bag

```
int bag::size(  ) const
// Postcondition: The return value is the number
// of integers in the bag.
{
    . . .
}
```

- Counts how many copies of a number occur

```
int bag::occurrences(int target) const
// Postcondition: The return value is the number
// of copies of target in the bag.
{
    . . .
}
```

Using the Bag in a Program

- Here is typical code from a program that uses the new bag class:

```
bag ages;

// Record the ages of three children:
ages.insert(4);
ages.insert(8);
ages.insert(4);
```

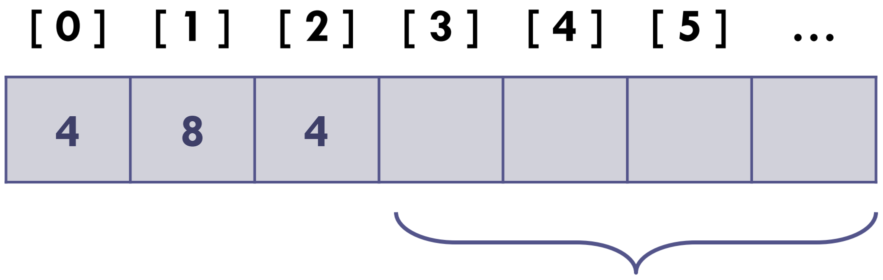
The Header File and Implementation File

- The programmer who writes the new bag class must write two files
- bag1.h, a header file that contains
  - documentation
  - class definition
- bag1.cpp, an implementation file that contains
  - the implementations of the bag's member functions

Implementation Details

- The entries of a bag will be stored in the front part of an array, as shown in this example
- The entries may appear in any order

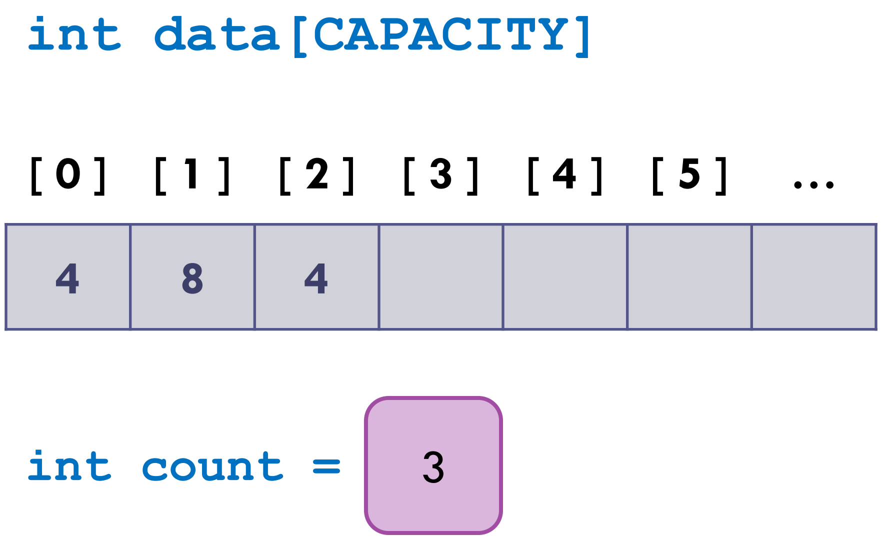
```
int data[CAPACITY]
```



We don't care what's in this part of the array

Implementation Details

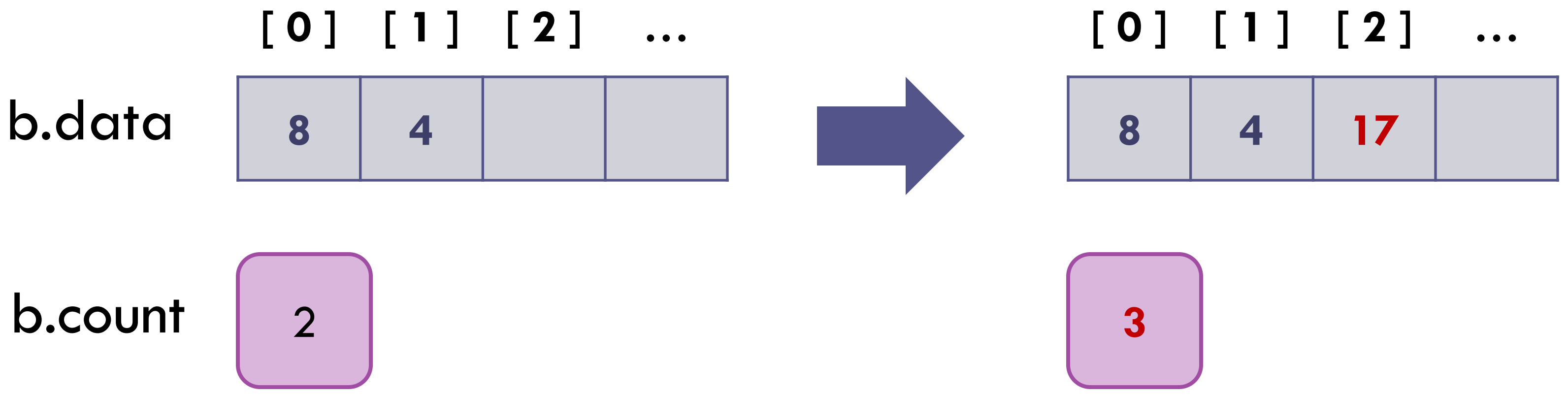
- We also need to keep track of how many numbers are in the bag



An Example of Calling Insert

```
void bag::insert(int new_entry)
```

```
bag b;
b.insert(17);
```



Pseudocode for bag::insert

- `assert(size( ) < CAPACITY);`

- Place new_entry in the appropriate location of the data array
- Add one to the member variable count

```
data[count] = new_entry;
count++;
```

or

```
data[ count++]  = new_entry;
```

Container Classes

- Classes designed to hold collections of objects
- Commonly provide services such as insertion, deletion, searching, sorting, and testing an item to determine whether it is a member of the collection
- Examples
  - Arrays
  - Stacks
  - Queues
  - Trees
  - Linked lists