

CONTAINER CLASSES

Problem Solving with Computers-II

<https://ucsb-cs24-sp17.github.io/>



Read the syllabus. Know what's required. Know how to get help.



CLICKERS OUT – FREQUENCY AB

Announcements

- Midterm next week (on Wed)
- Pa02 deadline moved to after midterm (04/28)
- Chapter 1 and 2 , PA 01, and all labs completed until now.

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.



Bags

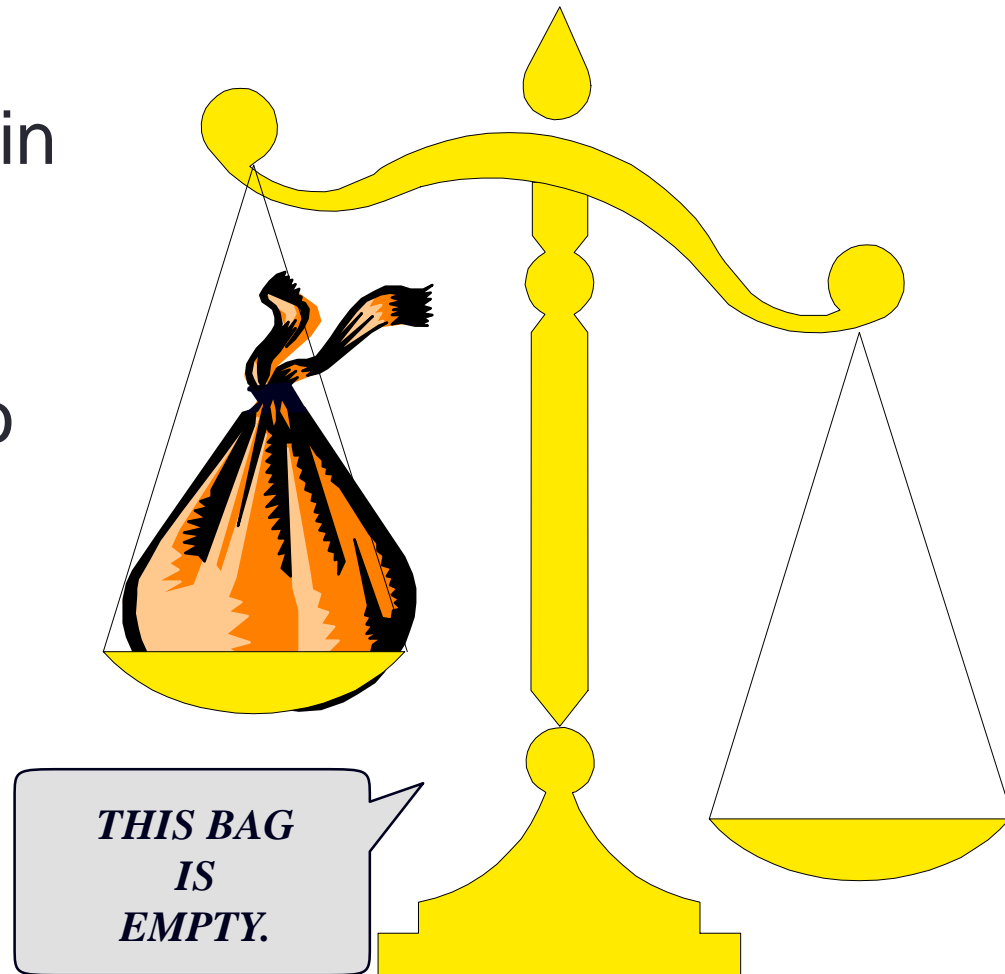
- For the first example, think about a bag.
- Inside the bag are some numbers.



Initial State of a Bag

- 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.

What questions should we ask to define the bag container class?



Defining the bag container class

- What questions should we ask to help define the bag container class?
- What can a user do with a bag of numbers



Inserting Numbers into a Bag

- Numbers may be inserted into a bag.



Inserting Numbers into a Bag

- Numbers may be inserted into a bag.



Inserting Numbers into a Bag

- Numbers may be inserted into a bag.
- The bag can hold many numbers.



Inserting Numbers into a Bag

- Numbers may be inserted into a bag.
- The bag can hold many numbers.



*THE 8 IS
ALSO IN
THE BAG.*



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.



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.



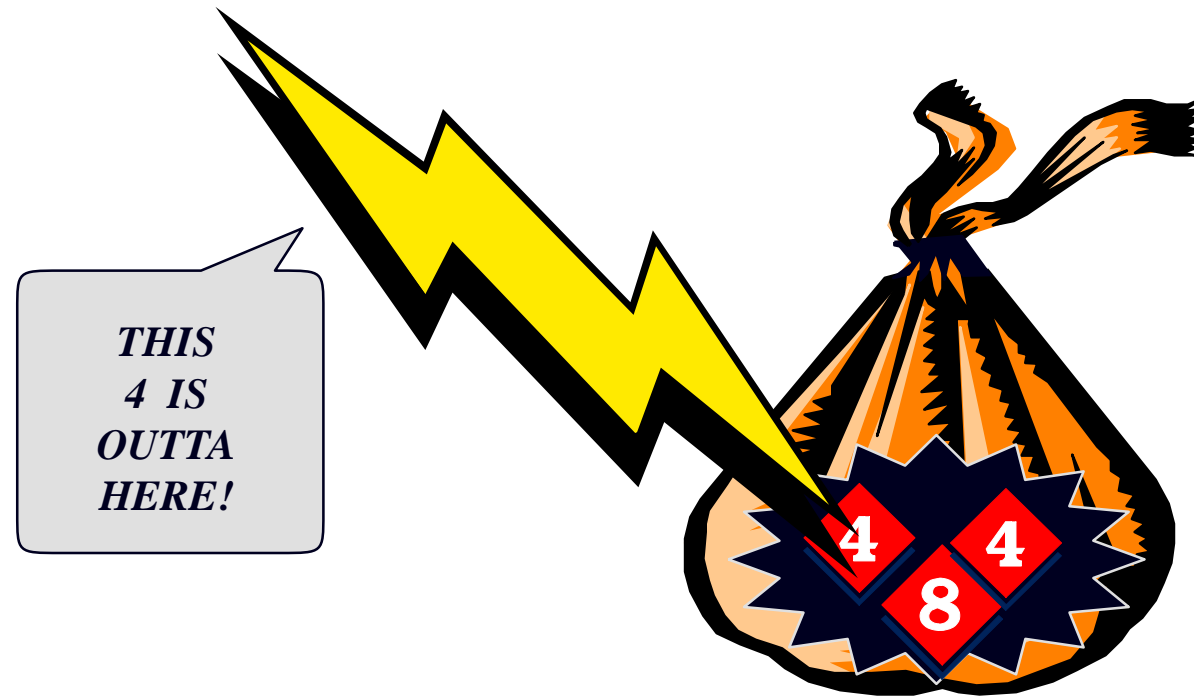
Examining a Bag

- We may ask about the contents of the bag.



Removing a Number from a Bag

- We may remove a number from a bag.



Removing a Number from a Bag

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



Which operations were defined so far on a “bag”

- A. Insert (possibly multiple instances of the same number)
- B. Count the number of occurrences of a number
- C. Remove a single occurrence of a number
- D. Some of the above
- E. All of the above

One more operation: Count how many numbers are in the bag in total

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



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.

Activity 1: Write the definition of the class

Operations defined so far -

- Create an **initial state**, which is an empty bag.
 - **Insert** a number
 - Count **occurrences** of a certain number
 - **Remove** a number from the bag.
 - Count **how many** numbers are in the bag.
-
- With your peer group write the definition of the bag class
 - Hints:
 1. What are the private data members?
 2. What are the methods?

```
class bag {  
    public:  
        void insert (int target);  
}
```

```
}
```

Demo

- Definition of the simple bag class: `simple_bag.cpp`
- Understand the use of static member variables by considering other options

Key take away: We used a static array to store the elements of a bag, why?

```
class bag
{
public:
    ...
private:
    int data[20];
    unsigned int count;
};
```

This limits the maximum number of elements!

We used a static const member variable for the capacity, why?

```
class bag
{
public:
    static const unsigned int CAPACITY = 20;
    ...
private:
    ✓ int data[CAPACITY];
    ✓ unsigned int count;
};
```

bag b1, b2, b3;
cout << b1.CAPACITY;

Don't forget to define
CAPACITY in the
implementation file!

cout << bag::CAPACITY;

bag b1, b2, b3;

How many copies of CAPACITY are created in memory by the above C++ statement?

```
class bag
{
public:
    static const unsigned int CAPACITY = 20;
    ...
private:
    int data[CAPACITY];
    size_t count;
};
```

A. Zero

B. One

C. Two

D. Three

E. Depends

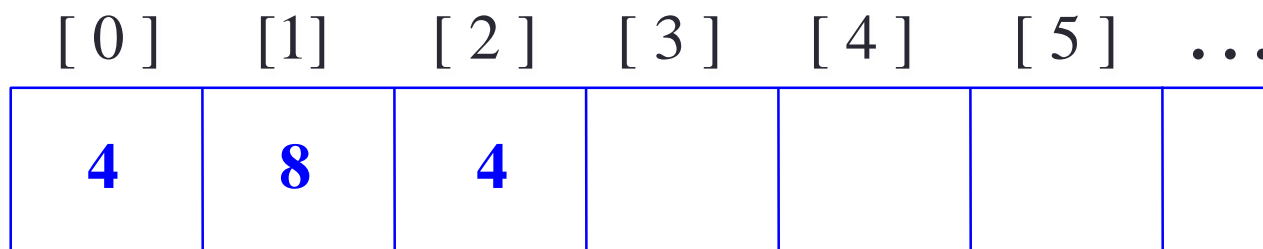
Specifying Class Invariant Implementation Details

Specifying Class Invariant

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



used - 1



Implementation Details

- The entries may appear in any order. This represents the same bag as the previous one. . .



[0]	[1]	[2]	[3]	[4]	[5]	...
4	4	8				



Implementation Details

- . . . and this also represents the same bag.



[0]	[1]	[2]	[3]	[4]	[5]	...
8	4	4				



Implementation Details

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

7 3



[0]	[1]	[2]	[3]	[4]	[5]	...
8	4	4				



Note: This is a key difference between the bag class and the sequence class (PA02) is that the for the sequence class, the order of elements is maintained after each deletion

An Example of Calling Insert

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

Before calling insert, we might have this bag b:



used



Precondition: Total no. of elements in the bag $<$ CAPACITY

Post condition: element inserted into bag

An Example of Calling Insert

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

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



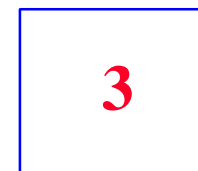
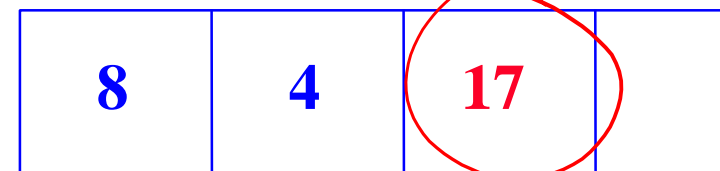
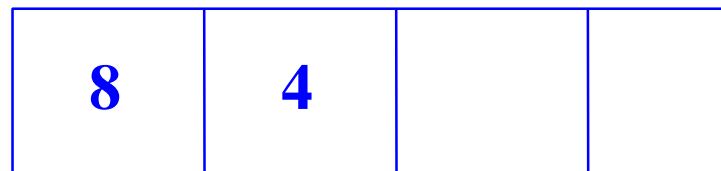
used



*What values will be in **b.data** and **b.count** after the member function finishes ?*

An Example of Calling Insert

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



Pseudocode for bag::insert

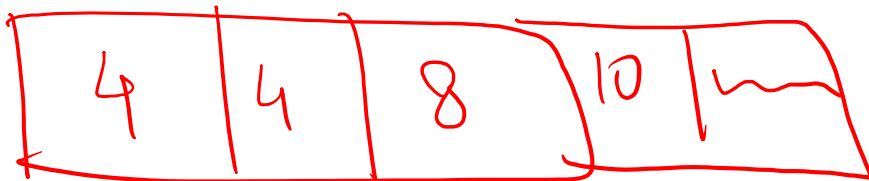
↪ assert(size() < CAPACITY);

✦ Place new_entry in the appropriate location of the data array.

✦ Add one to the member variable count.

data[used] = new_entry;
used++;

used = 4



What is the “appropriate location” of the data array?

// find the location of element (i)
// data[i] = data[used-1];
used - -

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++;
```

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;
```

Demo: Implementation of bag

- Definition of the simple bag class: `simple_bag.cpp`
- Implement the `insert`, `erase_one` and `count` methods
- Test with `simple_bag_test.cpp`
- Use the bag class to store the ages of members of a family, find each age and delete them (`bag_demo.cxx`)
- Consider the changes you have to make to the current code if we were to store a bag of elements of a different datatype.
- Compare with the other implementation (`bag1.cxx`) that uses a “flexible” data type for the elements of the bag.

What's new?

modify if
you want to
change the type
of elements stored
in the bag

```
class bag
{
  public:
    typedef int value_type;
    typedef std::size_t size_type;
    static const size_type CAPACITY = 30;
    bag( );
    bool erase_one(const value_type& target);
    void insert(const value_type& entry);
    size_type size( ) const { return used; }
    size_type count(const value_type& target) const;
  private:
    value_type data[CAPACITY];
    size_type used;
};
```

Other types of bags

- In the `simple_bag.cpp` example, we implemented a bag containing **integers**.
- The implementation in `bag1.cxx` allowed us to easily create a bag of **float numbers**, a bag of **characters**, a bag of **strings** . . .
- What was the key technique that allowed us to create a more flexible implementation?

Demo expected behaviour of pa02

Summary

- ❑ A container class is a class that can hold a collection of items.
- ❑ Container classes can be implemented with a C++ class.
- ❑ The class is implemented with a header file (containing documentation and the class definition) and an implementation file (containing the implementations of the member functions).
- ❑ Other details are given in Section 3.1, which you should read.

Next time

- Chapter 4: Pointers and dynamic arrays