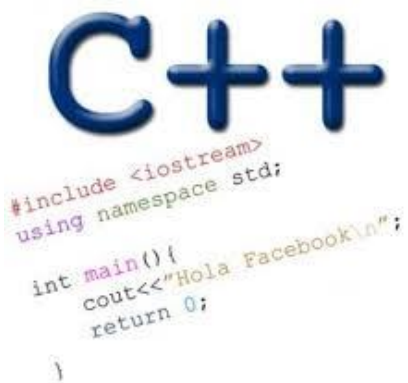
FRIEND FUNCTIONS
MAKEFILES (REVIEW)
GDB
CONTAINER CLASSES

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

- Extra open lab hours every week: Mondays 2pm to 4:00pm, 6:30pm to 8:00pm
- Reach out to your mentors!

Passing point objects as parameters (Review)

```
double distance(point p1, point p2);
```

//Precondition: p1 and p2 are point objects that have been initialized

//Post condition: returns the Euclidean distance between the two points

Would you implement the above function as a member function or a non-member function? Write your reason and discuss with your peer group.

A. Member function
B. Non-member function
C. Neither

Passing point objects as parameters (Review)

```
double distance(point p1, point p2);
```

//Precondition: p1 and p2 are point objects that have been initialized
//Post condition: returns the Euclidean distance between the two points

Which of the following is invoked when passing parameters to the distance function is (on line 2):

```
point s1(1,1), s2; //line 1
cout<<distance(s1, s2); //line 2
```

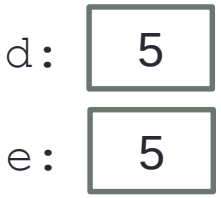
A. Default constructor
B. Default assignment operator
C. Default copy constructor

References in C++

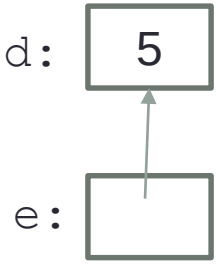
```
int main() {
  int d = 5;
  int &e = d;
}
```

Which diagram below represents the result of the above code?

A.



B.



C.

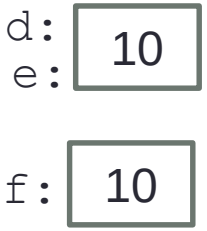
D. This code causes an error

References in C++

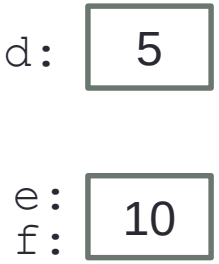
```
int main() {
  int d = 5;
  int &e = d;
  int f = 10;
  e = f;

}
```

How does the diagram change with this code?

A.



B.



C.

D. Other or error

Passing references as parameters

```
double distance(point &p1, point &p2);
```

//Precondition: p1 and p2 are point objects that have been initialized

//Post condition: returns the Euclidean distance between the two points

```
point s1(1,1), s2;
cout<<distance(s1, s2);
```

What is the benefit of passing references as parameters?

What are the potential dangers?

Operator overloading

In the previous class we overloaded the equality operator

==

bool operator ==(point p1, point p2); //function declaration

So we could use it in the following way:

```
double distance(const point & p1, const point &p2){
    if(p1 == p2)
      return 0;

}
```

Printing point objects to output stream

- By overloading the << and >> operators we could do the following :

```
point p(10, 10);
cout<<p;
```

And this….

```
point p;
cin>>p; //sets the x and y member variables of p based on user input
```

Demo

- New distance function
- Operator overloading and friend function wrap up
- Separate compilation with makefiles
- Debugging with gdb

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

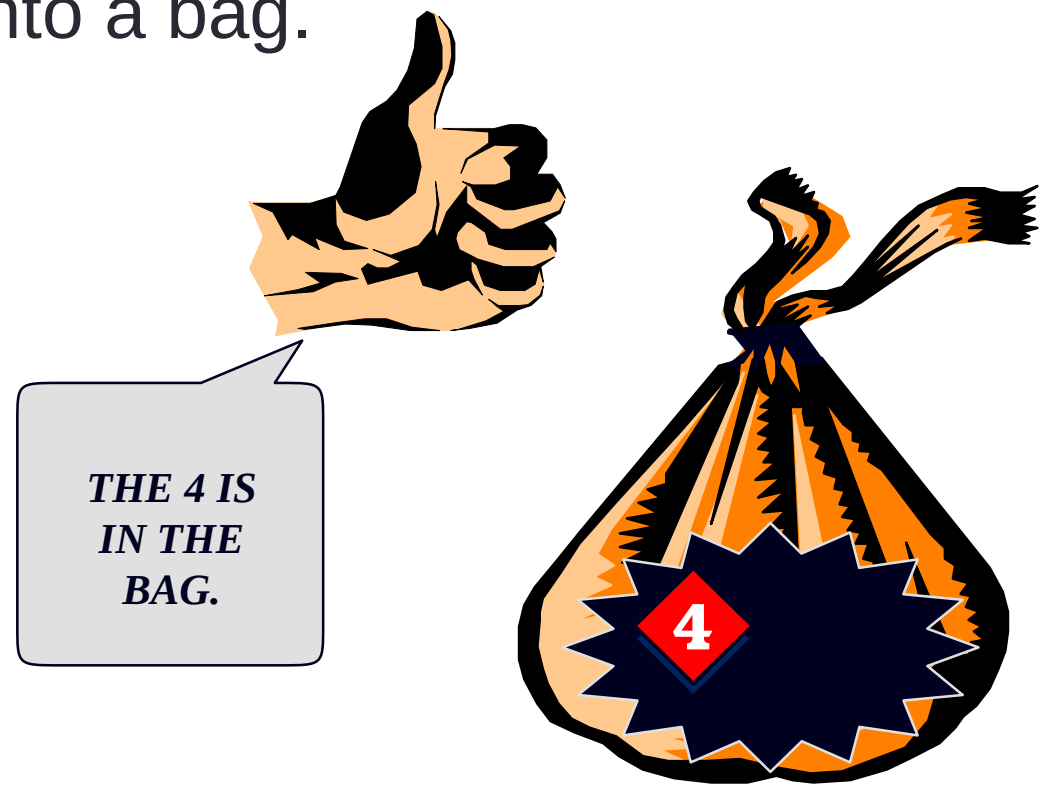
- When you first begin to use a bag, the bag will be empty.
- We count on this to be the **initial state** of any bag that we use.

Inserting Numbers into a Bag

- Numbers may be inserted into a bag.

Inserting Numbers into a Bag

- Numbers may be inserted into a bag.

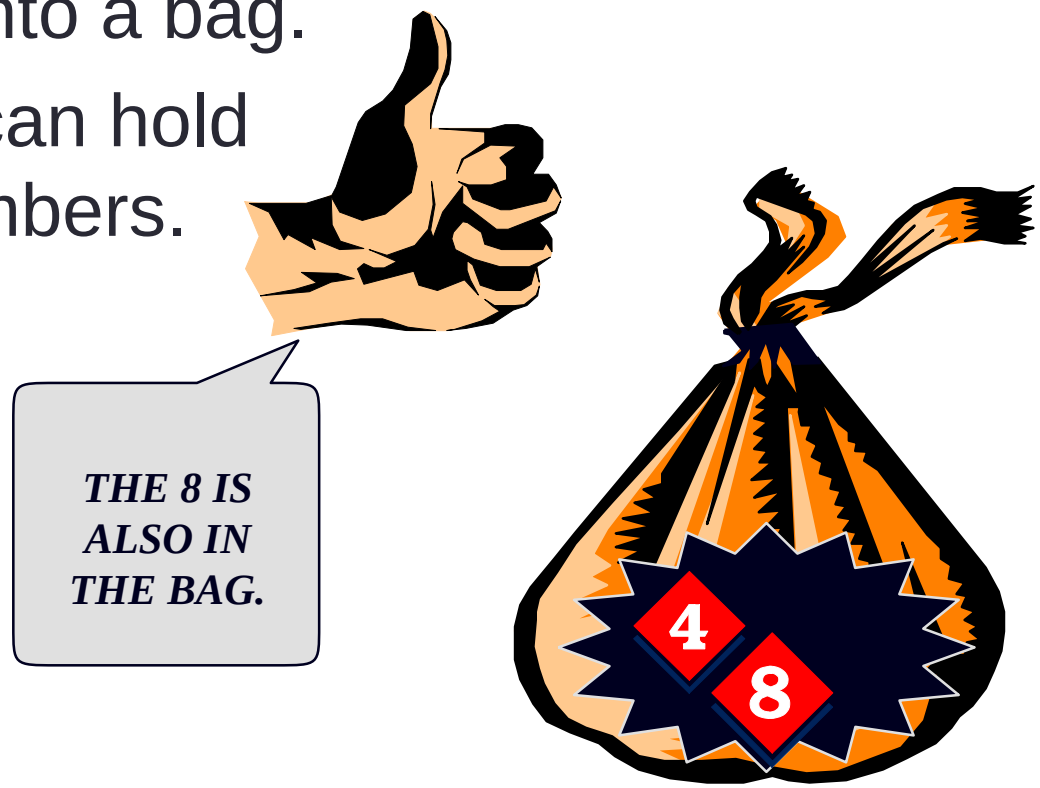


Inserting Numbers into a Bag

- 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.



Inserting Numbers into a Bag

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

Inserting Numbers into a Bag

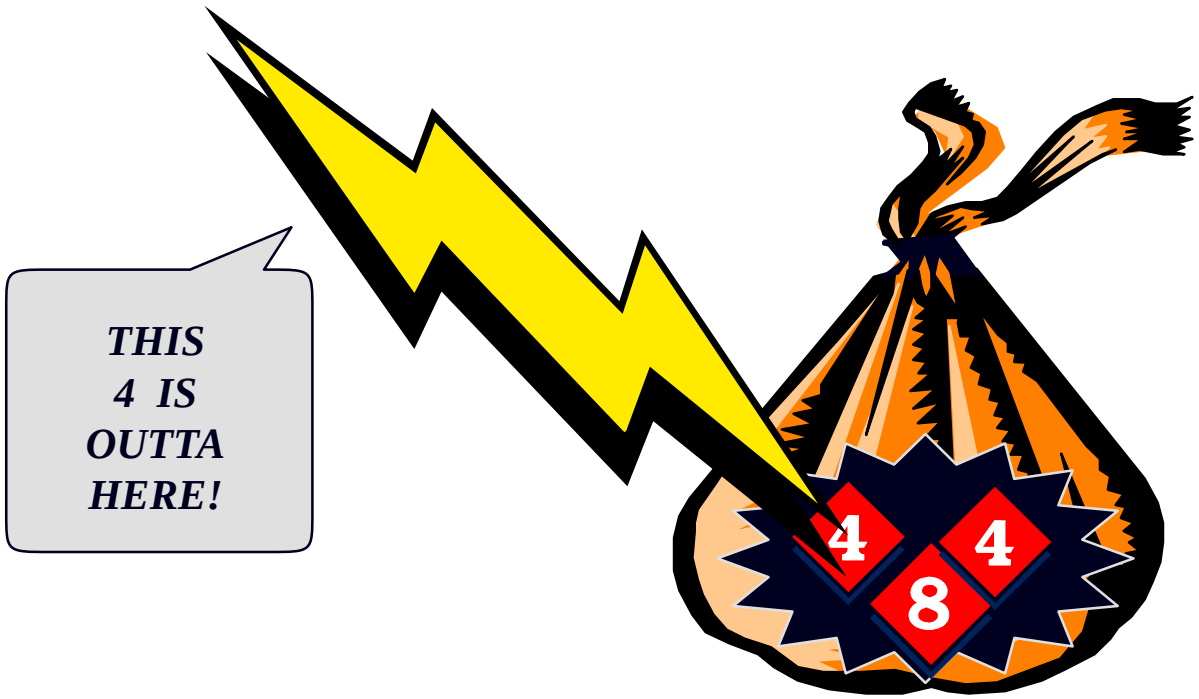
- Numbers may be inserted into a bag.
- The bag can hold many numbers.
- We can even insert the same number more than once.

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.

How Many Numbers

- 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.

The Bag Class

- C++ classes (introduced in Chapter 2) 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**

```
class bag
{
public:
        bag( );
        void insert(...
        void remove(...
        ...and so on
```

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);
```

Implementation Details

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

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

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.

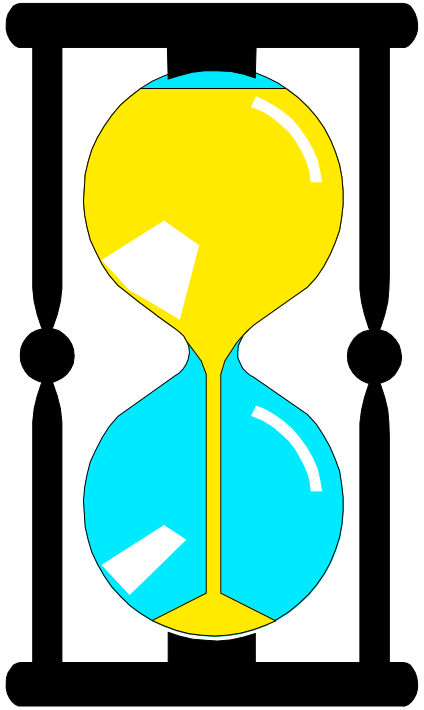
3

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

Note: A key difference between the bad class and the sequence class (PA02) is that the for the sequence class, the order of elements matters

An Exercise

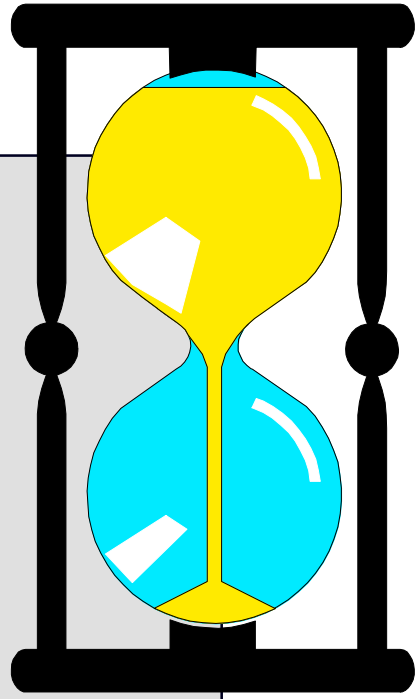
Use these ideas to write a list of private member variables could implement the bag class. You should have two member variables. Make the bag capable of holding up to 20 integers.



You have 60 seconds to write the declaration.

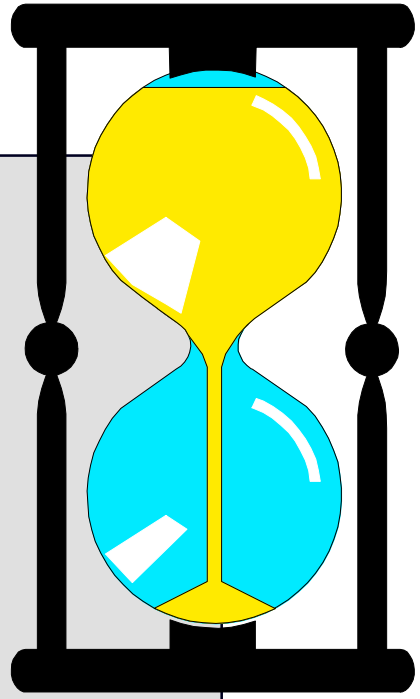
An Exercise

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



An Exercise

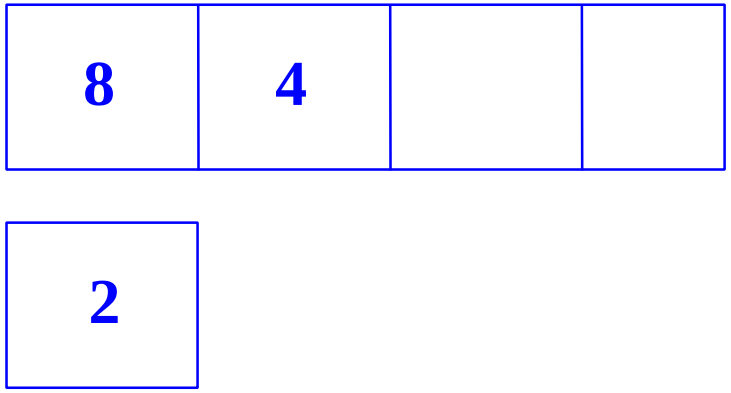
```
class bag
{
public:
    static const size_t CAPACITY = 20;
    ...
private:
    int  data[CAPACITY];
    size_t count;
};
```



An Example of Calling Insert

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

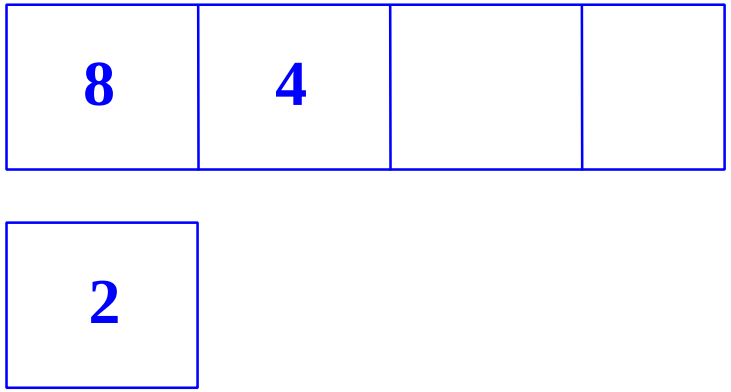
Before calling insert, we might have this bag b:



An Example of Calling Insert

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

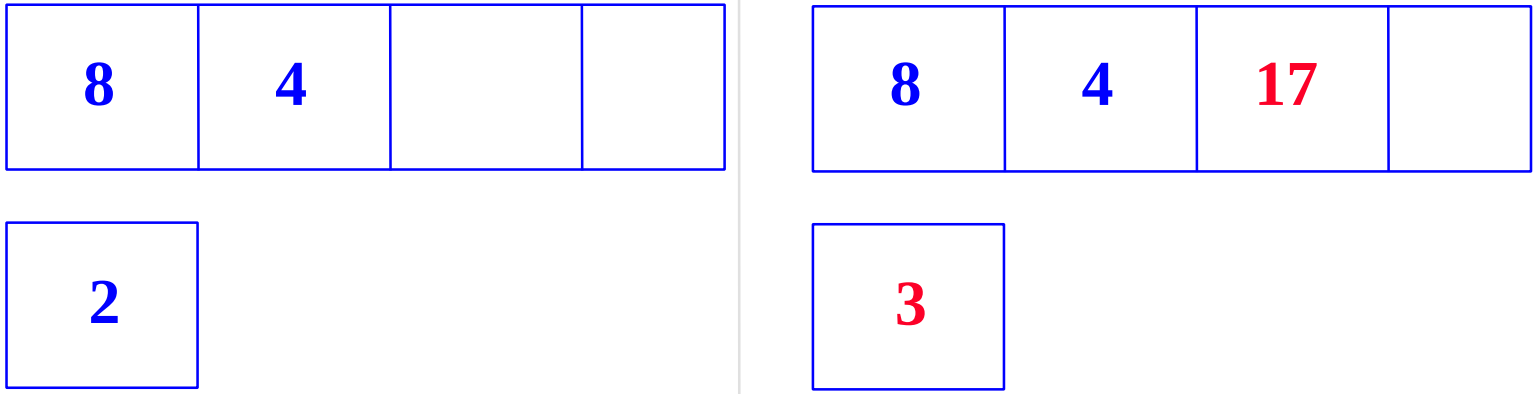
b.insert(17);



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.

What is the "appropriate location" of the data array ?

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;
```

The Other Bag Operations

- Read Section 3.1 for the implementations of the other bag member functions.
- Remember: If you are just **using** the bag class, then you don’t need to know how the operations are implemented.
- Later we will **reimplement** the bag using more efficient algorithms.
- We’ll also have a few other operations to manipulate bags.

Other Kinds of Bags

- In this example, we have implemented a bag containing **integers**.
- But we could have had a bag of **float numbers**, a bag of **characters**, a bag of **strings** . . .

Suppose you wanted one of these other bags. How much would you need to change in the implementation ?
Section 3.1 gives a simple solution using the C++ typedef statement.

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

- Container classes continued