BAG CLASS HOMEWORK

Implementing a Bag Class

Your assignment, **Implement the bag type (.cpp file)** and add to **main.cxx** to check all the functions!

The Bag class allows us to implement a bag of a given type using a dynamic array. What's neat about the bag class is we use "Type Definitions" in the .h file that allows us to change the type the class works with, with one change. Look at the .h file and notice the following two lines:

typedef int value_type;
typedef std::size_t size_type;

note that we define value_type as an integer type. That means *value_type x* is the same as saying *int x*. We use value type whenever we want to refer to this type throughout the class. Then when we want to change the class to a double we can just change this typedef statement in the .h file and the .cpp file and BAM our bag class now holds doubles.

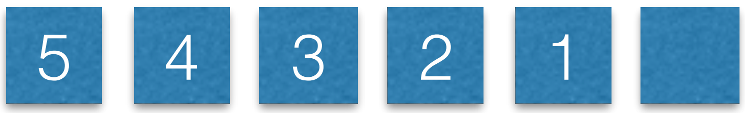
typedef double value_type;

A *size_t* is the same as an *unsigned int* in c++ which is just an integer that can only be positive (note, if you try to compare an int to a size_t type you will get a warning, to get rid of this warning, if you are comparing something (like a variable named x) to a size_t you can define it as size_type x; or unsigned int x).

Let's look at the .h file and see what we need to implement

A. Private Variables
*value_type *data;*
size_type used;
size_type capacity;

we have a pointer **data* that points to our dynamic array for our bag. We also have a variable used and a variable capacity. Let's say we have the following array with the first two elements set— this is what it might look like



data* points to the following array

used =5
***capacity**=6*

B. Implement Constructors and Destructors
bag(size_type initial_capacity= DEFAULT_CAPACITY)

You need to create *data to point to a dynamic array (using the *new*) of *value_type* with initial_capacity. Set used and capacity accordingly

bag(const bag& source)

Copy constructor. make sure you do a DEEP COPY meaning you create another dynamic array and set the array of our current bag instance to the array of the source instance. You also have to set used and capacity equal.

~bag(); //destructor

Destructor must free the dynamic memory allocated using *delete* command

C. Insert
void insert(const value_type& entry); //inserts into your bag

Insert the value into your dynamic array. Note that if you are at capacity, you need to reserve more space (i.e. make your bag bigger!).

D. Reserve
void reserve(size_type new_capacity);

Reserve is our first private function— We can make reserve a private function because it's only called from other member functions (i.e. it shouldn't be called from main). Make a new dynamic array with a greater capacity than your current bag. Then copy over your old bag into your new bag and then DELETE your old bag.

E. Erase_one and Erase
bool erase_one (const value_type& target);//erase 1 of the target value
erase 1 deletes 1 of the target value in the bag (in this case it returns true). Or it returns false if none exist in the bag
//if bag a has 1 2 3 4 in it
a.erase_one(3)
//bag a now has 1 2 4 in it.

size_type erase(const value_type& target)
erases all the values of target in the bag.
//if bag a has 2 2 3 4 in it
a.erase(2)
//bag a now has 3 4 in it

You should call erase_one from erase

F. Operator+= and Operator=
void operator +=(const bag& addend);
Operator += will do the following
//lets' say bag a has 1 2 3 and bag b has 4 5 6
a+=b;
//now bag a has 1 2 3 4 5 6, you can use ***insert*** here

void operator =(const bag& addend);
//let's say bag a has 1 2 3 and bag b has 2 3 4 5 6
a=b
//now bag a has 2 3 4 5 6

G. Size, Count, cout<<

```
size_type size() const {return used;}
size_type count (const value_type& target) const;
friend std::ostream& operator <<(std::ostream& outs, const bag& source);
```

Cout and size are implemented for you.
count takes in a target and gives you how many of a given type are in your instance

```
//if bag a has 4 5 3 4
cout<<a.count(4)
//2 will be coutted
```