```
#include <vector>
#include <cstddef>

// Constructor; creates and initializes an empty Bag of "capacity" size
template <class ItemType>
ArrayBag<ItemType>::ArrayBag(int capacity)
{
    items = new ItemType[capacity];

    myCapacity = capacity;
    itemCount = 0;

}

// Copy constructor; creates and initializes Bag from another Bag
template <class ItemType>
ArrayBag<ItemType>::ArrayBag(const ArrayBag& anotherBag)
{
    itemCount = anotherBag.getCurrentSize();

    myCapacity = anotherBag.getCapacity();

    items = new ItemType[myCapacity];

    for(int i = 0; i < itemCount; i++){
        items[i] = anotherBag.items[i];}


}

//destructor
template <class ItemType>
ArrayBag<ItemType>::~ArrayBag()
{
    delete [] items;
    items = NULL;
    itemCount = 0;
    myCapacity = 0;
}

// Assignment operator
template <class ItemType>
ArrayBag<ItemType>& ArrayBag<ItemType>::operator=(const ArrayBag<ItemType>& anotherBag)
{
    if(this == &anotherBag){


        itemCount = anotherBag.getCurrentSize();
        myCapacity = anotherBag.getCapacity();
        items = new ItemType[myCapacity];

        for(int i = 0; i < itemCount; i++){
            items[i] = anotherBag.items[i];}

        return *this;}
        return *this;
}


//Resizes the bag's capacity to newCapacity
//if the new size is larger, copy all bag contents
// we don't downsize a bag in HW2
template <class ItemType>
void ArrayBag<ItemType>::resize(int newCapacity)
{
    ItemType* newArray =  items;
```

```
    items = new ItemType[2 * myCapacity];

    for(int i = 0; i <= itemCount - 1; i++){
        items[i] = newArray[i];}

    newCapacity = myCapacity;
    delete [ ] newArray;
}

// Report whether the Bag is empty
// Return true if the Bag is empty (storing no Items);
// Return false if Items exist in the Bag
template <class ItemType>
bool ArrayBag<ItemType>::isEmpty() const
{
    return(itemCount == 0);
}

// Report whether the Bag is full
// Return true if the Bag is filled to capacity
// Return false if there is still room
template <class ItemType>
bool ArrayBag<ItemType>::isFull() const
{
    return(itemCount == myCapacity);    // STUB
}

// Give the Bag a new Item to store
// If Bag is full, double capacity and add newItem
// Else, Bag must add this Item to its Item array and update its numberOfItems
// If Bag is full after this, return true; else return false
template <class ItemType>
bool ArrayBag<ItemType>::add(const ItemType& newItem)
{
    if(!isFull()){

        itemCount++;
        items[itemCount - 1] = newItem;

        }
    else{

        resize(myCapacity);
        itemCount++;
        items[itemCount - 1] = newItem;}

    return true;


}

// Make the Bag act like an empty Bag again
template <class ItemType>
void ArrayBag<ItemType>::clear()
{
    itemCount = 0;
}


// Remove an Item from the bag
// If Item is not there, nothing changes and we return false
// Else, we fill in its spot in that Item array and count number of Items down
template <class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& anItem)
{
```

```
    if(contains(anItem) == true)

        for (int i = 0; i < itemCount; ++i){

            if (items[i] == anItem){

                items[itemCount] = anItem;
                itemCount = itemCount - 1;
            }
        }


    return false;// STUB
}

// Check if an Item is in the Bag
// Return true if it is in the Bag, and false if not
template <class ItemType>
bool ArrayBag<ItemType>::contains(const ItemType& anItem) const
{
    for (int i = 0; i < itemCount; ++i){
        if (items[i] == anItem)

            return true;
    }
    return false;
}

// Check how many times an Item is in the Bag
// return 0 if it's not there; otherwise,
// return the number of times it occurs
template <class ItemType>
int ArrayBag<ItemType>::getFrequencyOf(const ItemType& anItem) const
{
    int freque = 0;
    if(contains(anItem) == true){
        for(int i = 0; i <= itemCount; i++){
            if(items[i] == anItem){
                freque = freque + 1;}
        }}
    return freque;
}
// Make an output vector of Items from the bag (for checking)
template <class ItemType>
vector<ItemType> ArrayBag<ItemType>::toVector() const
{
    vector<ItemType> bagContents;

    for(int i = 0; i < itemCount; i++){
        bagContents.push_back(items[i]);
    return bagContents; }
    return bagContents;

}
```