Worksheet 0: Building a Simple ADT Using an Array
In Preparation: Read about basic ADTs.

In this worksheet we will construct a simple BAG and STACK abstraction on top of an array. Assume we have the following interface file "arrayBagStack.h"

```
# ifndef ArrayBagStack
# define ArrayBagStack
# define MAX_SIZE 100


# define TYPE int
# define EQ(a, b) (a == b)

struct arrayBagStack {
     TYPE data [MAX_SIZE];
     int count;
};

/* Interface for Bag */
void initBag (struct arrayBagStack * b);
void addBag (struct arrayBagStack * b, TYPE v);
int containsBag (struct arrayBagStack * b, TYPE v);
void removeBag (struct arrayBagStack * b, TYPE v);
int sizeBag (struct arrayBagStack * b);

/* Interface for Stack */
void pushStack (struct arrayBagStack * b, TYPE v);
TYPE topStack (struct arrayBagStack * b);
void popStack (struct arrayBagStack * b);
int isEmptyStack (struct arrayBagStack * b);

# endif
```

Your job, for this worksheet, is to provide implementations for the following operations.

```
/* Bag Implementation */
/************************************************************
*            void initBag(struct arrayBagStack * b)         *
* Parameters: struct arrayBagStack * b                      *
* Description: Bag implementation. Initializes count to 0 to  *
* correspond with the empty array.                          *
************************************************************/
void initBag(struct arrayBagStack * b) {
```

```
	b->count = 0; // == (*b).count = 0;	// Initialize count to 0.
	//https://stackoverflow.com/questions/2575048/arrow-operator-usage-in-c
}

/************************************************************
*		void addBag(struct arrayBagStack * b, TYPE v)	*
* Parameters: struct arrayBagStack * b, TYPE v		*
* The first parameter is the array, the second parameter is	*
* the element (in this case an integer, since TYPE int) to be *
* added to the array.					*
* Description: This function is responsible for adding an	*
* element to the bag and incrementing the count afterwards.	*
************************************************************/
void addBag(struct arrayBagStack * b, TYPE v) {

	b->data[b->count] = v;			// Add the element to the array.
	b->count++;				// Increment count.

}

/************************************************************
*	int containsBag(struct arrayBagStack * b, TYPE v)	*
* Parameters: struct arrayBagStack * b, TYPE v		*
* The first parameter is the array, the second parameter is	*
* the element (in this case an integer, since TYPE int) we	*
* are checking the array for.				*
* Description: This function uses a for loop to go through	*
* each element of the array to locate a user-specified value. *
* The function returns true (1) if the user's value is in the *
* array. Otherwise, it returns false.			*
************************************************************/
int containsBag(struct arrayBagStack * b, TYPE v) {

	// For loop to go through each element.
	// int i = 0;
	// While i < count (number of elements in array)
	// Increment i
	// Terminate loop when i is no longer < count.
	for (int i = 0; i < b->count; i++) {
		if (EQ(b->data[i], v))
			return 1;
	}
	return 0;				// int function must return value.
}

void removeBag (struct arrayBagStack * b, TYPE v) {
for (int i = 0; i < b->count; i++) {
	if (EQ(v, b->data[i])) {
		for (int j = i; j < b->count; j++) {
			if (j < 99) {
				b->data[j] = b->data[j + 1];
			}
			else {
				b->data[j] = 0;
			}
			b->count--;
			return;
```

```
            }
        }
}
}

/*************************************************************
*            int sizeBag(struct arrayBagStack * b)           *
* Parameters: struct arrayBagStack, * b                      *
* Description: This function returns the size of the array.  *
* Since count is incremented each time an element is added to *
* the array, count contains the total number of elements in the *
* array at any given time. Therefore, when this function is  *
* called, we return the count, which is the size of the array. *
*************************************************************/
int sizeBag(struct arrayBagStack * b) {

        return b->count;                        // == (*b).count;

}

/* Stack Implementation */
/*************************************************************
*      void pushStack(struct arrayBagStack * b, TYPE v)      *
* Parameters: struct arrayBagStack * b, TYPE v               *
* The first parameter is the array, the second parameter is  *
* the element (in this case an integer, since TYPE int) the  *
* user wants to push onto the stack.                         *
* Description: This function pushes the user's specified     *
* value onto the system stack. This code is borrowed from the *
* CS261 - Abstract Data Types video lecture.                 *
*************************************************************/
void pushStack(struct arrayBagStack * b, TYPE v) {

        addBag(b->data, v);

}

/*************************************************************
*      type topStack(struct arrayBagStack * b)               *
* Parameters: struct arrayBagStack * b,                      *
* The first parameter is the array, the second parameter is  *
* the element (in this case an integer, since TYPE int) the  *
* user wants to push onto the stack.                         *
* Description: This function returns the value found on the  *
* top of the stack.                                          *
*************************************************************/
TYPE topStack(struct arrayBagStack * b) {
        assert(!isEmptyArray(b));   // Check to make sure array isn't empty.
        // The stack is a last-in-first-out, so the last element added would be on
        // the top. Therefore, we want to return the last element added to the array:
        return b->data[b->count - 1];
}

void popStack(struct arrayBagStack * b) {
        assert(!isEmptyArray(b));
        // Because the stack is a last-in-first-out structure, "popping" the stack would
        // remove the last element added.
```

```
        // This would also decrease the number of elements in the array by one, therefore:
        b->count--;   // Decrement the count by one, since one element has been removed.

}

int isEmptyStack(struct arrayBagStack * b) {

        // CS 261 - Abstract Data Types Lecture Video
        return(!b->count);
        //return (sizeBag(b->data) == 0); // == return(!b->count);??
        //return (arraySize(b->data)==0)
}
```