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

struct arrayBagStack {
TYPE data [100];
int count;
};

void initArray(struct arrayBagStack * b);
void addArray (struct arrayBagStack * b, TYPE v);
int containsArray (struct arrayBagStack * b, TYPE v);
void removeArray (struct arrayBagStack * b, TYPE v);
int sizeArray (struct arrayBagStack * b);

void pushArray (struct arrayBagStack * b, TYPE v);
TYPE topArray (struct arrayBagStack * b);
void popArray (struct arrayBagStack * b);
int isEmptyArray (struct arrayBagStack * b);
# endif
```

Your job, for this worksheet, is to provide implementations for all these operations.

```
void initArray (struct arrayBagStack * b){

        b->count = 0;

 }
```

/* Bag Interface Functions */

```
void addArray (struct arrayBagStack * b, TYPE v) {

	if(b->count < 100) {
		b->data[b->count] = v;
		b->count++;
	} else
		printf("Error, bag Is full\n");
}

int containsArray (struct arrayBagStack * b, TYPE v){

	int i;

	for(i = 0; i < (b->count); i++) {

		if (EQ(b->data[i], v))
			return 1;	// return 1 for true if found in array
	}

	return 0;	// return 0 If value is not found in the array
}

void removeArray (struct arrayBagStack * b, TYPE v) {

	int i;
	int index = -1;

	//loop through array to find the Index of value
	for(i = 0; i < (b->count); i++) {

		if (EQ(b->data[i], v))
			index = i;
	}

	if(index != -1) {
		b->data[i] = b->data[(b->count) - 1];
		b->count--;
	} else
```

```
		printf("Error, value not In array\n");
}

int sizeArray (struct arrayBagStack * b) {

	return b->count;
}



/* Stack Interface  Functions */

void pushArray (struct arrayBagStack * b, TYPE v) {

	if(b->count < 100) {
		b->data[b->count] = v;
		b->count++;
	} else
		printf("Error stack Is full\n");
}

TYPE topArray (struct arrayBagStack * b) {

	if(b->count > 0)
		return b->data[(b->count) - 1];
	else
		return 0;	// return 0 If array Is empty


}

void popArray (struct arrayBagStack * b) {

	if(b->count > 0) {
		b->data[(b->count) - 1] = 0;
		b->count--;
	} else
		printf("Error, stack Is empty\n");
}
```

```
int isEmptyArray (struct arrayBagStack * b) {
        if(b->count == 0)
                return 1;    // return 1 If stack Is empty
        else
                return 0;    // return 0 If stack Is not empty
}
```

A Better Solution…

This solution has one problem. The arrayBagStack structure is in the .h file and therefore exposed to the users of the data structure. How can we get around this problem? Think about it…we'll return to this question soon.

A solution to this problem would be to provide a "new" function for the data structure. So a function called newArrayBagStack() would create a new structure for the user. However, in using this function, the user wouldn't actually be creating a new arrayBagStack structure but would simply be returned a pointer to an already existing struct that is "hidden" in the .c file. The user can always declare pointers without needing to know what the pointers are pointing to.