Worksheet 0 (Solution): 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.

```
/*
 *  arrayBagStackStack.c
 *  arrayBagStackStack
 *
 *  Copyright 2010 Oregon State Univ. All rights reserved.
 *
 */

#include "arrayBagStack.h"
```

```
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>

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


/* Bag Interface */

/* This function allocates space for an arrayBagStack structure! */
struct arrayBagStack * createArray()
{
        struct arrayBagStack * b = malloc(sizeof(struct arrayBagStack));

        return b;
}

void initArray (struct arrayBagStack *b)
{
        b->count = 0;
        /* Why don't we init the array contents?  It's actually not necessary since we only
access elements
         * up to count, meaning, we will only attempt to access (ie. read) elements of the
array which
         * have necessarily been assigned since we increment count only when we've
         * assigned a new value */

}


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

        b->data[b->count] = v;
        b->count++;
}

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 0;

}

void removeArray (struct arrayBagStack * b, TYPE v)
{
        int i = 0;
        int j;

        assert(containsArray(b,v));  /* error if they try to remove something not in the
array */

        /* Find the index after the element*/
        while(!EQ(b->data[i], v))
                                        i = i + 1;
        /* Slide elements to fill the gap */
        for(j=i;j<b->count-1;j++)
                b->data[j] = b->data[j+1];

        b->count = b->count-1;

}

int sizeArray (struct arrayBagStack * b)
{
                return b->count;
}



/* Stack Interface */

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

        addArray(b,v);
}

TYPE topArray (struct arrayBagStack * b)
{
        assert(!isEmptyArray(b));
        return b->data[b->count-1];
}
```

```
void popArray (struct arrayBagStack * b)
{
        assert(!isEmptyArray(b));
        b->count--;
}


int isEmptyArray (struct arrayBagStack * b)
{
        return( !b->count);
}


void printArray(struct arrayBagStack *b)
{
        int i;

        for(i=0; i < 100; i++)
                printf("Array [ i ] = %d \n", b->data[i]);
}
```

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.