Worksheet 0: Building a Simple ADT Using an Array

In Preparation: Read about basic ADTs.

Group 11 week 2 participants:

Tatyana Vlaskin
Danny Mejia
Katherine Jensen

Our group conversation over these worksheets can be found on Piazza, "(Worksheet_Group11) Group 11: Week 2".

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.

IMPLEMENTATION:

```
#include "arrayBagStack.h"
#include<assert.h>
#include<stdlib.h>
#include<stdio.h>
struct
arrayBagStack{
TYPE data[100];
int count; };
```

// WE HAVE NOT REACHED CONSENSUS, 2 IMPLEMENTATIONS ARE PROVIDED

IMPLEMENTATION#1:

```
void initArray (struct arrayBagStack * b){
   b->data = malloc(100 * sizeof(TYPE)); // setting capacity to 100
   assert(b->data != 0);
   b->size=0;
   b->capacity=100;
}
```

IMPLEMENTATION #2

```
void initArray(struct arrayBagStack *b){
        //allocation of  space for the array
        b->data = malloc(100 * sizeof(TYPE)); // setting capacity to 100
          int i;
          for(i=0; i<100; ++i){
                    b->data[i]=0;// not sure if we need this because count is
zero
          }
          b->count=0;// setting count to 0
          }
```

// WE HAVE NOT REACHED CONSENSUS, 2 IMPLEMENTATIONS ARE PROVIDED

IMPLEMENTATION#1:

```
/* Bag Interface Functions */
void addArray (struct arrayBagStack * b, TYPE v) {
   if (b->size <= b->capacity-1)
      b->data [b->size] = v;
   b->size+=1;
}
```

IMPLEMENTATION#2:

```
void addArray(struct arrayBagStack *b, TYPE v){
          if (b->size <= b->capacity-1)
```

```
                b->data[b->count] = v;// element is added to the array
                b->count++;// any time an element is added, we increment count
}

// WE HAVE NOT REACHED CONSENSUS, 2 IMPLEMENTATIONS ARE
PROVIDED
IMPLEMENTATION#1:

int containsArray (struct arrayBagStack * b, TYPE v){
    int i;
    for (i=0; b->data[i] != 0 ; i++)
        v++;
    return v;
 }
IMPLEMENTATION#2
int containsArray(struct arrayBagStack *b, TYPE v){
 int i;
 for(i=0;i<b->count; ++i){
     if(v==b->data[i]){ // if there in an element in the array that is equal to the
v
             return 1; //OR WE CAN RETURN V
     }
 }
 return 0;// if element is not in the array, zero will be returned.
}


void removeArray(struct arrayBagStack *b, TYPE v){
          int i, j;
          if(containsArray(b, v)){// before we remove the element from the array
we need to make sure it exists there
                    for(i=0; i < b->count; ++i){// loop through the elements
                              if(v == b->data[i]){ //
                                        for(j = i; j < b->count-1; ++j){
                                                  b->data[j] = b->data[j+1]; //shift elements to
fill the spot that was removed                                  }
                                        break;
                              }
                    }
                    b->count--;//decrement count
          }
}}


int sizeArray (struct arrayBagStack * b) {
    return b->size;
```

```
}

/* Stack Interface  Functions */
// WE HAVE NOT REACHED CONSENSUS, 2 IMPLEMENTATIONS ARE
PROVIDED
IMPLEMENTATION#1:

void pushArray (struct arrayBagStack * b, TYPE v) {
   if (b->size <= b->capacity-1)
      b->data[b->size] = v;
   b->size+=1;
}

IMPLEMENTATION#2
//push is equivalent to add an Array, so we can use addArray function
void pushArray(struct arrayBagStack *b, TYPE v){

          addArray(b,v);
}


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

// WE HAVE NOT REACHED CONSENSUS, 2 IMPLEMENTATIONS ARE
PROVIDED
IMPLEMENTATION#1:

void popArray (struct arrayBagStack * b) {
   int i;
   for (i = v;  i <= b->size ; i++)
      b[i] = b[i+1];
        b->size--;
}
IMPLEMENTATION #2

void popArray(struct arrayBagStack *b){
          assert(!isEmptyArray(b));
          b->data[b->count-1] = 0;// last element in the array if removed
          b->count--;//count is decremented
}
// WE HAVE NOT REACHED CONSENSUS, 2 IMPLEMENTATIONS ARE
PROVIDED
```

IMPLEMENTATION#1:

```
int isEmptyArray (struct arrayBagStack * b) {
   int i;
   for (i=0; i <= b->size ; i++) // IS SIZE SAME THING AS COUNT
      if (b->data[i] == 0)
         return i;
}
```

IMPLEMENTATION #2

```
int isEmptyArray(struct arrayBagStack *b){
          if(0 == b->count){ // If there is no elements, count is zero.
                    return 1;
          }
          return 0; // if array is not empty return 0
```

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.

Question was answered using information on the following website: **http://stackoverflow.com/questions/1154709/how-can-i-hide-the-declaration-of-a-struct-in-c**

In the header file include:
typedef struct arrayBagStack Point;
As the complier sees this it knows that there is a struct called arrayBagStack.
At the same time it knows that there is a pointer Point that can refer to a arrayBagStack.
This will hide Information about how struc looks like what members it contains and how big it is.

Some members of the group thought namespace would be useful here but C doesn't use that.