Chapter 3: Array-Based Implementations

Vocabulary

Write out a description of each vocabulary term for reference later.

Term	**Description**

Concepts

Core Methods of an ADT

Data structures are structures that store data. This means that they store data, and usually will have some public-facing functions to deal with *adding* data, *finding* data, and *removing* data, though some structures might have more sophisticated functionality as well.

Add functions are usually called “push”, and implementing a “push_back” function in an array would be easy – simply add the new data to the next available space in the array.

However, implementing an insert function would be more work – you don’t want to overwrite existing data, so first you must move everything over and then insert the new data at the position specified.

Arrays and common errors

Once you’re using arrays in a program, it can be easy to have your program crash. You’ve probably experienced the “out-of-bounds” error before.

For an array of size *n*, the minimum valid index is:

For an array of size *n*, the maximum valid index is:

When writing a data structure that uses arrays, we need to protect against these errors – our code shouldn’t cause other peoples’ software to crash!

For example, we would need to add error checking to the following functions:

Function	`Push( TYPE newthing )`
Error checks	Make sure that the array is not full (if a static array) Resize the array if it is full before an insert (if a dynamic array)
Function	`Get ( int index )`
Error checks	Make sure the index is >= 0 Make sure the index is < SIZE

Design

When creating a data structure that uses an array, we have to make a design decision: Do we allow gaps? Or should we keep all elements consecutive? And what does it matter?

Allowing gaps

0	1	2	3	4	5
A	B		C	D	

Everything is consecutive

0	1	2	3	4	5
A	B	C	D		

If we are allowing spaces, then any time we want to add more data to the structure we have to find an empty spot. This means having a loop like this:

```
int i;
for ( i = 0; i < size; i++ )
{
    if ( IsEmpty( arr[i] ) )
    {
        break; // use this index
    }
}

if ( i != size )
{
    // Insert at this position
    Insert( arr[i], data );
}
```

In the worst case scenario, we might search the entire array only to find no available spaces! This can be time consuming, especially if we have a lot of data. Imagine having millions of pieces of data and having to search through the entire array every time we want to add something – it is inefficient!

If, however, we keep our array elements consecutive, we just need to keep track of the next available index. Having an integer member variable like `m_itemCount` will work. When the structure is first created, m_itemCount is initialized to 0. As we insert new items, it will look like this:

```
if ( m_itemCount != size )
{
    // array isn't full, insert here.
    arr[ m_itemCount ] = data;
    m_itemCount++;
}
```

Which essentially is instantaneous; we don't need to loop through the entire array. This is much more efficient. We can also then tell immediately if the array is full or not: if the `size` of the array is 5 (valid indices are 0 through 4), and `m_itemCount` is 5, then the array is full.

What is m_?

Rachel prefixes **private** member variables with "m_".
In some cases online, you might also see these prefixed with just "_".

Your coding style is up to you, but there may be a specific coding style that is used on the job!

For the ADT Bag, what functions are created?

1.

2.

3.

4.

5.

6.

7.

8.

Why should data members be private?

When should member functions be marked as `const`?

What should the constructor function initialize?

Testing

When implementing a data structure, it is useful to create a *tester program* to test out the functionality, prior to putting it in an actual project. There are various types of tests we can do. On page 104, the tests written must be verified by human eyes to tell if they were successful. Later on, we will talk about **Unit Tests** and how we can write programs that validate functionality for us automatically.

What is a function stub? What is it used for?

When writing tests, should you try to pass in invalid data to see how the structure responds?