ECE 2574 Data Structures and Algorithms

Meeting 2: ADT Bag

Today we will take a look at our first ADT, how to define an interface for it in C++, and discuss possible implementations. Along the way we will get an introduction to CMake and testing using Catch.

Today's Schedule:

- ADT Bag
- Bag Definition using a Templated Class.
- Testing the Bag using Catch and CMake

Warmup #1

Consider the ADT Bag as defined on page 21 of the text **without** the `toVector()` method ...

A *Bag* holds a finite number of objects of the same type, not necessarily distinct, with no particular ordering.

- `construct()`: construct an empty bag
- `destroy()`: destroy the bag and any contents
- `add( Item )`: add an `Item` to the bag, returns true on success, else false
- `remove( Item )`: remove a single instance of `Item` from the bag, returns true on success, else false
- `isEmpty()`: returns true if the bag has no contents, else false
- `getCurrentSize()`: returns the number of items in the bag as an integer
- `clear()`: removes all items in the bag
- `getFrequencyOf( Item )`: the number of times item appears in the bag
- `contains( Item )`: returns true if at least one Item is in the bag, else false

Warmup #1

Consider the ADT Bag as defined on page 21 of the text **without** the `toVector()` method. What could you do with such an ADT (check all that apply)

- Determine the percentage of entries in a bag equal to a given entry (74%) CORRECT
- Sort the entries according to some criteria (23%)
- List each entry in the bag with its frequency of occurance (57%)

Warmup #2

What is the primary mechanism for implementing ADTs in C++?

Classes (89% got this correct)

Classes are the primary mechanism for implementing ADTs in C++

A type is a concrete representation of a concept.

For example, the type float approximates a real number

A class is a user-defined type that extends the built-in types.

For example, the class Bag models the concept of the ADT Bag.

Classes provide many advantages for implementing ADTs.

- They can hide implementation details via private.
- They provide a means of forcing the ADT interface to be used.
- They enable type-checking on complex concepts.
- They assist with assertion checking and maintaining constraints.

All of this helps to keep the object, an instance of the type, in a **well-defined state**.

Templates

A bag of integers and a bag of strings are basically the same. All that is really required for the type of bag objects is that they can be copied (or moved) and there is a test for equality. We can declare a generic class as

```
template<typename T> class Bag;
```

Exercise (see starter code on website)

Consider a simplified version of the Bag ADT (without toVector). In `bag.hpp` define a templated class for the simplified Bag ADT.

- The class is named Bag.
- Use std::size_t rather than int as the type for sizes.
- Pass entries by constant reference
- Take care with const correctness.

Implement stubs for these methods in the same header file, but after the class declaration.

Warmup #3

Describe an example of a test you might write to check an implementation of the ADT Bag.
Examples:

- Allocate a Bag of Ints, test that its size is 0
- Allocate a Bag if Ints, add some to it, check size, contains, etc.

The most common misconception is that an application is not really a test. You want to know your ADT works separate from using it in an application.

Next Actions

- Read CH pp. 31-37 (C++ classes), this should largely be a review
- Take warmup before 8am on Friday 8/26.