CS 261 – Data Structures

Abstract Data Types
What is an abstraction?

Merriam Webster

1. remove, separate
2. to consider apart from application to or association with a particular instance
3. to make an abstract of: summarize
4. to draw away the attention of

Wikipedia

Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon, typically to retain only information which is relevant for a particular purpose. For example, abstracting a leather soccer ball to the more general idea of a ball retains only the information on general ball attributes and behaviour, eliminating the characteristics of that particular ball
Container Abstractions

• Over the years, programmers have identified a small number of different ways of organizing and operating on collections of data
• These container abstractions are now the fundamental heart of the study of data structures

Examples: bag, stack, queue, set, map, etc
Three Levels of Abstraction

There are three levels of abstraction that we will consider in the study of data structures:

• Specification/Interface: Properties and behaviors (what)
• Application: How it’s used (why)
• Implementation: the various implementations in a particular library (how)

Can you describe the three levels of abstraction of the stack ADT?
Stack ADT

Specification/Interface View

```c
initStack( );
pushStack(val);
valType topStack( );
popStack( );
bool isEmptyStack( );
```

Properties: A Stack is a collection that has the property that an item removed is the most recently entered item [LIFO]

In C, we’ll describe the interface in the .h files with function prototypes and comments
**Stack ADT**

Implementation View

```c
void pushArray(struct arrayStack *stk, double val) {
    arrayAdd(stk->data, val);
}
```

```c
int arrayIsEmpty(struct arrayStack *stk) {
    return (arraySize(stk->data) == 0)
}
```

In C, our implementation will go in `.c` files

Note that an ADT can have MANY implementations using several different data structures
Stack ADT

Application View

Given an expression ((2+3) * 4) , can you describe how you would use a stack to ensure that the ( parens ) are properly balanced?
(See explanation in Chapter 6)

(2 + 3)) // not balanced
(2 – 3 ( // not balanced
(( 5 + 6) * 2) // balanced
(2+3))

Error: attempt to pop from empty stack
(2-3)

Error: Done processing tokens and the stack is not empty
((5+6) * 2) BALANCED!
Classic ADTs

Simple collections:
• Bag
• Ordered bag

Arranged by position:
• List (Indexed)

Ordered by insertion:
• Stack
• Queue
• Deque

Ordered by removal:
• Priority Queue

Unique Elements
• Set

Key/Value Associations
• Map or Dictionary
Array Implementation of the Stack ADT

Example Usage:

```c
struct arrayStack myStack;
initArray (myStack);
pushArray(&myStack, 5);
```
The Bag ADT

**Application:** Used in applications where you need to maintain an unordered collection of elements (duplicates allowed), without needing to know how it is organized. Very commonly used ADT. (e.g. shopping cart)

**Interface/Behavior Specification:**

- Add (val)
- bool Contains (val)
- Remove (val)

**Implementation:** Worksheet 0: Bag Interface
Your Turn

Worksheet 0: array implementation of Bag & Stack

Example Usage:

```c
struct arrayBagStack myBag;
initArray(&myBag);
addArray (&myBag, 5);
addArray (&myBag, 23);
if(containsArray (&myBag, 24))
    printf("Bag contains a 24!\n");
```