CS 261 - Spring 2012

Abstract Data Types
By
Tim Budd
Sinisa Todorovic
Container Classes

• A small number of different ways to organize data

• These abstractions are our focus

• Examples: Stack, Queue, Set, Map, etc.
Three Levels of Abstraction

• ADT - Abstract Data Type, language independent -- **What is it?**

• Interface - in a particular library of containers -- **How to use it?**

• Implementation - in a particular library – **How it works?**
Abstract Data Type View

• Data type described in a language-independent way

• Properties are true regardless of the names given to operations in library

• E.G., A Stack is a collection where an item removed is the most recently entered item
Metaphors

- ADT view are often described by metaphors (e.g., stack of plates). Easy to understand.
The Interface View

- Gives specific names to operations
- In C, interface is defined by .h files

```c
struct stack;
void initStack (struct stack * stk);
void pushStack (struct stack * stk, double val);
double topStack (struct stack * stk);
void popStack (struct stack * stk);
int isEmptyStack (struct stack * stk);
```
Additional Information

• The interface view gives only names

• Must also attach meanings
  – E.g., LIFO properties of stack, etc

• May attach expected execution times
  – E.g., want push and pop to be constant time
int stackIsEmpty (struct stack * stk) {
    return dyArraySize(stk->data) == 0;
}
The Classic ADTs

- Bag, Ordered Bag - simple collections
- Stack, Queue, Deque - ordered by insertion
- Set - unique elements, fast test
- Map (Dictionary) - key/value associations
- Priority Queue - ordered by importance
BAGs
BAG as ADT

• **Problem:** Need to maintain an *unordered* collection of elements

• **Operations:**
  – Insert
  – Remove
  – Contains

• **Behavior:** time of operations is unimportant
BAG as ADT

• **Specific requirements:**
  – Time of insertion is important

• **Solution:** Provide abstract interface
BAG interface

• Provide functions for operations

  addBag (container, value)
  testBag (container, value)
  removeBag (container, value)
  sizeBag (container, value)
Stack as ADT

• **Problem**: Maintain a collection of elements in Last-In, First-Out format

• **Operations**:
  – Add an element to Stack
  – Remove an element from Stack
  – Contains
The Classic Implementation

- Arrays and Dynamic Arrays (Vectors)
- Linked Lists
- Binary Trees, Balanced Trees
- Heaps
- Hash Tables
- Skip Lists
- Etc etc etc
Worksheet 0
Worksheet 0

• Make implementations of a simple BAG and STACK using an array in C

• We will do (a part of) these together, in class
First look at C - interface file

# define TYPE double
# define EQ(a, b) (a == b)

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

void initBag (struct arrayBag * b);  … etc
First function - initialize

void initBag (struct arrayBag * b)
{

}
Add to bag

void addBag (struct arrayBag * b, TYPE v)
{
}

Test for contains

int containsBag (struct arrayBag * b, TYPE v)
{
}
}
Remove from bag

void removeBag (struct arrayBag * b, 
    EleType v)
{
}
Return size of collection

```c
int sizeBag (struct arrayBag * b) {
}
```
How about stack?

void pushStack (struct arrayBag * b, TYPE v)
{

}
Test for stack empty

int isStackEmpty (struct arrayBag * b)
{

}
Top of stack

EleType topStack (struct arrayBag * b)
{
}

}
void popStack (struct arrayBag * b)
{
}

Pop top element