CS 261 – Data Structures

Abstract Data Types (ADTs)

Container Classes

- A few 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
  - **Defines what it is.**

- Interface - in a particular library of containers
  - **Defines how to use it.**
  - **Behavior constraints.**

- Implementation - in a particular library
  - **Specifies how it works.**

Abstract Data Type View -- What

- Every data type can be described in a language-independent way
- Properties are true regardless of the names given to operations in a library
- Example: A Stack is a collection of data items where the last added item must be removed first (LIFO)

Metaphors

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

The Interface View – How to Use

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

- Example: Stack interface
  - initStack = Initialize the stack
  - pushStack = Add an element to the stack
  - popStack = Read and remove an element from the stack
  - topStack = Read the top element of the stack
  - isEmptyStack = Check if the stack is empty

Interface – How to Use in C

- Example: Stack interface in 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);
```

Interface – Behavior Constraints

In addition to names, the interface also specifies:

- Meaning of ADT
  - E.g., LIFO properties of stack, etc.
- Behavior constraints
  - E.g., want Push and Pop to be constant time

The Classic ADTs

- **Bag, Ordered Bag**: simple collections of data
- **Stack, Queue, Deque**: ordered by insertion
- **Priority Queue**: ordered by removal
- **List**: captures sequential dependencies between data
- **Tree, Graph**: captures complex dependencies of data
- **Set**: has all unique elements => fast test
- **Map** (Dictionary): key/value associations

Bag ADT

Bag ADT

- **Definition**: Maintains an **unordered** collection of data elements

- **Operations**:
  - Initialize the bag
  - Add/Remove a data element
  - Check if the bag contains a given element
  - Check the size of the bag

- **Behavior**: specific requirements on time of operations

Bag Interface

- Provide functions for operations on a bag, effectively hiding implementation

```
initBag (container);

addBag (container, value);

containsBag (container, value);

removeBag (container, value);

sizeBag (container);
```

Worksheet 0

Implement the Bag ADT using the static **array** type in C

Interface .h File for a Bag

```
# define TYPE double
# define MAX_SIZE 100
struct Bag {
   TYPE data[MAX_SIZE];
   int size;
};
/* function prototypes */
void initBag (struct Bag *b);
void addBag (struct Bag *b, TYPE val);
int containsBag (struct Bag * b, TYPE val);
void removeBag (struct Bag * b, TYPE val);
int sizeBag (struct Bag * b);
```

Implementation -- Add an Element to Bag

```
void addBag (struct Bag * b, TYPE val){

   /*check if the bag was initialized*/
    assert(b != NULL);


    /*check if there is enough memory space*/
    if (b->size >= MAX_SIZE) return;

    b->data[b->size] = val;

    b->size++; /*adding an element increases size*/

}
```

Test if Bag Contains an Element

```
int containsBag (struct Bag *b, TYPE val){
   assert(b != NULL); /*check if b was initialized*/
   int i = b->size - 1; /*index of the last element*/

   /* elements in the bag are not ordered, so we
      need to exhaustively search for the element*/
   while (i >= 0) {
     if (b->data[i] == val) return 1;
     i--;
   }
   return 0;
}
```

Remove an Element from Bag

```
void removeBag (struct Bag * b, TYPE val) {

    /* Insert your code */

    /* Note that you cannot have "holes" in array */


}
```

Remove an Element from Bag

```
void removeBag (struct Bag * b, TYPE val) {
   assert(b != NULL); /*check if b was initialized*/
   int i = b->size-1; /*index of the last element*/
   while (i >= 0) {/*exhaustive search*/
      if (b->data[i] == val){ /*found it*/
         /*swap last with current to avoid the gap*/
         b->data[i] = b->data[b->size - 1];
         b->size--; /*the size decreases by one*/
         return;/*removes one occurrence*/
      }
      i--;
   }
}
```

Complexity? O(n)

Next Lecture

- Dynamic array -- introduction