

Geometric Algorithms and Data Structures

Part II: Linked Structures, Scan-Line Algorithms, and Search Trees

Jim Fix

Building your own data structures

In some of your programs, you will discover a need to store a collection of items in some single, object— a *data structure*.

Most programming languages come with a sequenced structure: typically either an array or a list, or both. Classes also provide such a structure, each instance holds a fixed-size collection of data items in its *fields*.

A data structure conforms to a particular interface—this defines its *abstract data type*. It may be built using some other structure — its *implementation*. For example, an efficient priority queue can be built as a binary heap, using an array.

Many languages provide data structures in libraries as well. However, at times, you might build the structure from scratch.

Example: Stacks, Queues, ...

```
class Stack {  
    Stack(...) { ... }  
    void push(int v) { ... }  
    int pop() { ... }  
    int top() { ... }  
}
```

```
class Queue {  
    Queue(...) { ... }  
    void enqueue(int v) { ... }  
    int dequeue() { ... }  
    int head() { ... }  
}
```

Example: Priority Queues, Dictionaries...

```
class PQueue {
    PQueue(...) { ... }
    void insert(int v) { ... }
    int extractMax() { ... }
    int max() { ... }
}
```

```
class Dictionary {
    Dictionary(...) { ... }
    void insert(int key, Object value) { ... }
    int delete() { ... }
    Object find(int k) { ... }
}
```

Today: look at *link-based* implementations of data structures, including the *ordered dictionary* type.

Array-based Stack

We can build a Stack class using an array.

```
int[] elts;  
int topAt;
```

```
Stack(int size) {  
    elts = new int[size];  
    topAt = 0;  
}
```

```
int top() {  
    return elts[topAt-1];  
}
```

```
...
```

Stack built as an Array (cont'd)

```
...  
void push(int v) {  
    elts[topAt] = v;  
    topAt++;  
}  
  
int pop() {  
    topAt--;  
    return elts[topAt];  
}
```

Any such Stack object has a size limit.

Link-Based Data Structures

A *link-based* data structure is one that uses object references (*links*) to organize a collection of data in a program.

It stores each data item in an object (*a node*). That node contains references/links to other nodes in the structure.

Different node connection structures are used depending on the collection's *interface* (its supported methods), etc. Examples include binary trees like the heap, and linked lists.

A *Linked List Stack*

Idea: Store each stack value in an object that knows

- the value itself
- the object of the item “underneath” it in the stack (i.e. next item down)

```
class ListStackElt {
    int value;
    ListStackElt next;

    ListStackElt(int v, ListStackElt n) {
        value = v;
        next = n;
    }
}
```

The bottommost stack item’s object has no next object. It is the *null* object. We keep track of the entire stack by remembering the object of the top of the stack.

The Details of ListStack

```
class ListStack {  
  
    ListStackElt topElt;  
  
    ListStack(int size) { topElt = null; }  
  
    void push(int v) { topElt = new ListStackElt(v,topElt); }  
  
    int pop() {  
        int v = top();  
        topElt = topElt.next;  
        return v; }  
  
    int top() { return topElt.value; }  
}
```