Elementary Data Structures

Alan G. Labouseur, Ph.D.
Alan.Labouseur@Marist.edu
Algorithms :: Elementary Data Structures

Algorithms are general recipes for solving problems not specific of any language or platform.

To study algorithms, we need something for them to act upon. That brings up **Data Structures**.

As with algorithms themselves, data structures present the same challenges:

**Challenges:**
- Correctness
- Efficiency
- Applicability
But before we get to those, consider the array data type.

This is an array of length 8 indexed from 0 to 7. We could fill this array with data, for example ...
Algorithms :: Arrays

We could fill this array with eight James Bond movie names.

```c
string bondFilms[7] = { };
bondFilms[0] = “Dr. No”;
bondFilms[1] = “From Russia with Love”;
```

```
<table>
<thead>
<tr>
<th>Dr. No</th>
<th>From Russia with Love</th>
<th>Goldfinger</th>
<th>On Her Majesty’s Secret Service</th>
<th>The Spy Who Loved Me</th>
<th>Moonraker</th>
<th>For Your Eyes Only</th>
<th>GoldenEye</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>[1]</td>
<td>[2]</td>
<td>[3]</td>
<td>[4]</td>
<td>[5]</td>
<td>[6]</td>
<td>[7]</td>
</tr>
</tbody>
</table>
```

In this example, we have an array of string (text) values. We can have arrays of other data types, including pointers.
Algorithms :: Arrays

We could fill this array with pointers to objects representing eight James Bond movie names.
Algorithms :: Lists

Elementary Data Structures
- Lists
- Stacks
- Queues

Q: Does an array for a list?

A: It certainly can.
Especially if we know in advance how many items we want to store in the list.

But what if we need to be more flexible?
Linked Lists

For flexibility, we need a linked list.

It’s a **list**... of **linked** objects.
Linked Lists

We need to know the **head** or **start** of the list. We also need to denote that the last item has no **Next** pointer.
Linked Lists

Pointers are really addresses in memory.

Film
Dr. No
Next
0x3A28213A

Film
From Russia with Love
Next
0x6339392C

Film
Goldfinger
Next
0x7363682E

Film
On Her Majesty's Secret Service
Next

Film
The Spy Who Loved Me
Next

Film
Moonraker
Next

Film
For Your Eyes Only
Next

Film
GoldenEye
Next
NULL

from the great XKCD

Head or Start of the list
0x3A28213A
Linked Lists

Pointers are really addresses in memory. And those hexadecimal addresses are really just shorthand for binary.
Linked Lists

We can use linked lists to build elementary data structures.
Elementary Data Structures

- Lists
- Stacks
- Queues

Stack

\[ \text{top} \rightarrow \]

- R
- A
- C
- E
- C
- A
- R

Stack
Stack

A stack is an abstract data type that supports the following operations:

- **Push** - add an element to the top of the stack
- **Pop** - remove an element from the top of the stack
- **isEmpty** - check to see whether or not the stack is empty
Stack

Operations:
- Push
- Pop
- isEmpty

Film
On Her Majesty’s Secret Service

Next

Film
Dr. No

Next

Film
From Russia with Love

Next

Film
Goldfinger

Next

Film
On Her Majesty’s Secret Service

Top
Stack

Operations:
• Push
• **Pop**
• isEmpty

\[ m = \text{stack.pop()} \]
Stack

Operations:
• Push
• Pop
• isEmpty

\[ m = \text{stack.pop()} \]
Stack

Operations:
• **Push**
• Pop
• isEmpty

```python
stack.push(GoldenEye)
```
Stack

Operations:
• **Push**
• Pop
• isEmpty

```javascript
stack.push(GoldenEye)
```
Algorithms :: Queues

Elementary Data Structures

- Lists
- Stacks
- Queues

Queue

| R | A | C | E | C | A | R |

head

tail
A queue is an abstract data type that supports the following operations:

- **Enqueue** - add an element to the back of the queue
- **Dequeue** - remove an element from the front of the queue
- **isEmpty** - check to see whether or not the queue is empty
Queue

Operations:
- Enqueue
- Dequeue
- isEmpty

![Diagram of a queue with films]

*Head*:
- Dr. No
- From Russia with Love
- Goldfinger
- On Her Majesty's Secret Service

*Tail*:
- NULL
Queue

Operations:
• Enqueue
• Dequeue
• isEmpty

m = queue.dequeue()
Queue

Operations:
• Enqueue
• Dequeue
• isEmpty

\[ m = \text{queue.dequeue()} \]

Head

<table>
<thead>
<tr>
<th>Film</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>From Russia with Love</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Film</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goldfinger</td>
<td>Next NULL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Film</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>On Her Majesty's Secret Service</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Tail
Queue

Operations:
- **Enqueue**
- Dequeue
- isEmpty

```python
queue.enqueue(GoldenEye)
```
Queue

Operations:
- **Enqueue**
- Dequeue
- isEmpty

```
queue.enqueue(GoldenEye)
```
Algorithms :: Performance Characteristics

What’s the “Big Oh” of these operations?

• Lists
  ‣ add
  ‣ remove

• Stacks
  ‣ push
  ‣ pop

• Queues
  ‣ enqueue
  ‣ dequeue