College of Information Studies
University of Maryland Hornbake Library Building College Park, MD 20742-4345

Data Structures

Week 7

INFM 603

Muddiest Points

- Nested loops (4)
- Reading code (2)
- Getting started on a program (1)
- Syllabus (1)

The Key Ideas

- Structured Programming
- Modular Programming
- Data Structures
- Object-Oriented Programming

Arrays

- A set of <u>elements</u>
  - For example, the number of days in each month
- Each element is assigned an <u>index</u>
  - A number used to refer to that element
    - For example, x[4] is the <u>fifth</u> element (count from zero!)
  - Arrays and iteration work naturally together
- "Constructor" allocates space
  - var myArray = new Array(5); // all unitialized
  - var myArray = [42, 17, , 22, 1]; // partially initialized

Array Example

```
// allocate five-element Array (indexed 0..4)
var n = new Array(5);

// assign values to each element of Array n1
for (var i=0; i<n.length; i++) {
  n[i] = i;
}

// output index and value of each element
for (var i=0; i<n.length; i++) {
  document.writeln(i + ": " + n[i]);
}
```

Data Structures

- Constant
  - Names given to unchanging values (for readability)
- Scalar
  - Single-valued item (int, float, boolean)
- Object
  - Multi-valued item, mixed data types [+methods]
- Array
  - Integer-indexed set of objects (usually of one type)
- Associative array (“hash table”)
  - Object-index set of objects (usually of one type)

Associative Arrays in JavaScript

```
var myArray = new Array();
myArray['one'] = 1;
myArray['two'] = 2;
myArray['three'] = 3;

// show the values stored
for (var i in myArray) { // skips unitialized
  alert('key is: ' + i +
        ', value is: ' + myArray[i]);
}
```

Common Uses of Arrays

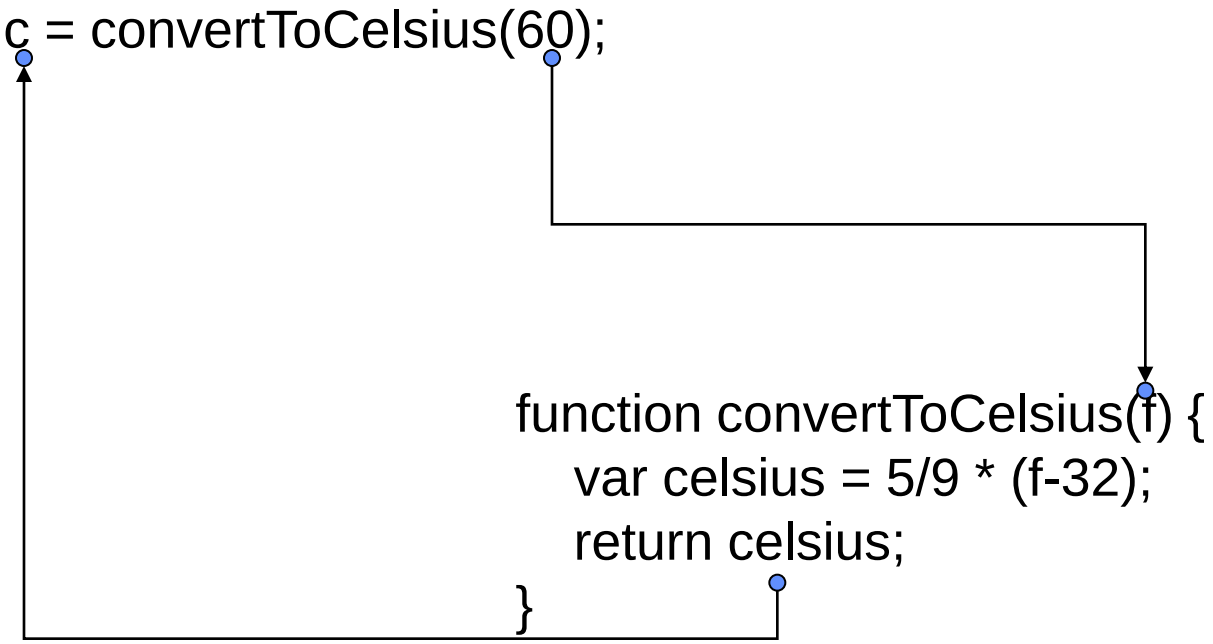
- Iterative computation
- Queue (FIFO)
- Stack (LIFO)

Functions

- Reusable code for complex "statements"
  - Takes zero or more values as "parameters"
  - Returns <u>at most one</u> value as the "result"

```
function convertToCelsius(f) {
   var celsius = 5/9 * (f-32);
   return celsius;
}
```

```
var f = 60;
c = convertToCelsius(f);
```



Uses of Functions

- Compactness
  - Minimizing duplicative code
- Modular programming
  - Abstraction
  - Reusability
- Avoid “side effects” for modular programming

Scope of a Variable

- In JavaScript, *var* "declares" a variable

var mystery;	create a variable without defining its type
var b = true;	create a boolean *b* and set it to true
var n = 1;	create an integer *n* and set it to 1
var s = "hello";	create a string *s* and set it to "hello"

- Variables declared in a function are "local"
  - Function parameters are implicitly declared (local)
  - Same name outside function refers to **different** variable

- All other variables are "global"

Parameter Passing

- Scalars are copied
  - "Pass by value"
- Arrays (and all objects) pass "by reference"
  - The values in the array are **not** copied
    - Be careful to make "side effects" explicit
  - No need to return the **same** reference
- Functions can also be passed as parameters
  - Unchangable, so "by reference" = "by value"
- Returned values work the same way

Recursion

- A function can call itself
  - Local variables are different each time
- Every invocation of the function must end
  - There must be a path that ends the recursion
  - That path must eventually be taken
  - The usual way to do this is an initial **if** statement
- Never essential
  - But sometimes more elegant than iteration

Binary Search with Recursion

```
function binarySearch(theArray, key, low, high) {
  var middle;
  if (low>=high) {                          // Safety check!
    if (key==theArray[low]) {
      return low;
    } else {
      return -1;
    }
  } else {
    middle = Math.floor((low+high)/2); // Explicit!
    buildOutput( theArray, low, middle, high );
    if (key<=theArray[middle]) {        // Equality!
      return binarySearch(theArray, key, low, middle);
    } else {
      return binarySearch(theArray, key, middle+1, high);
    }
  }
}
```

Using JavaScript with Forms

HTML:

```
<form name="input" action=" ">
   Please enter a number:
   <input size="10" value=" " name="number"/>
</form>
<form name="output" action=" ">
   The sum of all numbers up to the number above is
   <input size="10" value=" " name="number" readonly="true"/>
</form>
```

Reads in a value from the first form
(*eval* method turns it into a number)

JavaScript:

```
var num = eval(document.input.number.value);
document.output.number.value = 10;
```

Changes the value in the second form

HTML Form Element Types

- Textarea (multiple lines)
- Input
  - Text (single line)
  - Password (like text, but masked)
  - Hidden (like text, but not displayed at all)
  - Button
  - Checkbox (multiple selection)
  - Radio (single selection)
- Select (dropdown list)

see `http://www.w3schools.com/html/html_forms.asp` for examples

Linking Forms to Functions

- Events:
  - Actions that users perform

- An “event handler” is triggered by an event
  - onClick: the user clicked on the item
  - onMouseover: the mouse moved onto the item
  - onMouseout: the mouse moved off of the item

Referring to Form Content

```
<form name=years>
  <b>Please enter your age</b>
  <input type=text name=box />
</form>
…
var age = document.years.box.value;
```

```
<form action = " ">
   <p>Enter integer search key<br />
   <input id = "inputVal" type = "text" />
</form>
…
var inputVal   = document.getElementById("inputVal");
var searchKey  = inputVal.value;
```

Design Tips

- Protect against unexpected values
  - Test the value of **all** user input
  - Test the value of critical function parameters
- Verify that every loop will **always** terminate
  - Include a bailout condition, and report it
- Always test for conditions explicitly
  - Trap unexpected conditions with the final else

Programming Tips

- Attention to detail!
  - Careful where you place that comma, semicolon, etc.
- Don’t get cute with the logic or the layout
  - Reflect the structure of your problem clearly
  - Use standard “design patterns”
- Write a little bit of code at a time
  - Add some functionality, make sure it works, move on
- Debug by viewing the “state” of your program
  - Print values of variables using document.writeln();

Documentation Tips

- Reflect your pseudocode in your code
  - Use meaningful variable names
  - Use functions for abstractable concepts
    - And name those functions well
  - Use comments to fill remaining gaps
- Add a comment to identify each revision
  - Give author, date, nature of the change
- Waste space effectively
  - Use indentation and blank lines to guide the eye

Why Program?

- Data manipulation
- Simulation
- Control
  - Interaction
  - Embedded

Before You Go

On a sheet of paper, answer the following (ungraded) question (no names, please):

What was the muddiest point in today's class?