- PHP Expressions and Operators
- Similar to those in C++ / Java / Perl
- Be careful with a few operators
    - / in PHP is always floating point division
        - To get integer division, we must cast to int

```
$x = 15;
$y = 6;
echo ($x/$y), (int) ($x/$y), "<BR />";
    > Output is  2.5 2
```

    - Mixed operands can produce odd results
        - Values may be cast before comparing

- To compare strings, it is better to use the C-like string comparison function, strcmp()
- Other string functions are listed in Sebesta Table 9.3
- Even the == operator has odd behavior in PHP when operands are mixed
- Ex: Consider comparing a string and an int
    - Any non-numeric PHP string value will "equal" 0
    - Any numeric PHP string will equal the number it represents
    - Ex: Consider comparing a string and a boolean
        - Regular PHP string value will "equal" true
        - "0" string will equal false
    - This behavior is consistent but confusing to the programmer and is probably best avoided

- An additional equality operator and inequality operator are defined
  - === returns true only if the variables have the same value and are of the same type
    - > If casting occurred to compare, the result is false
  - !== returns true if the operands differ in value or in type
- Precedence and associativity are similar to C++/Java
  - See http://us2.php.net/manual/en/language.operators.precedence.php

- PHP Control Structures
- Again, these are similar to those in C++ / Java
    - **if**, **while**, **do**, **for**, **switch** are virtually identical to those in C++ and Java
    - PHP allows for an alternative syntax to designate a block in the if, while, for and switch statements
        - Advantage to this syntax is readability
            - > keywords vs. brackets
            - > Now instead of seeing a number of close braces, we see different keywords to close different types of control structures

- A nice feature of PHP is that the "control" resulting from a control structure is maintained even when you exit back to html mode
    - Thus, in <?php you can branch / loop etc.
    - You can then exit php ?> and format in straight html
- PHP also has the foreach loop
    - Similar to the Java for loop for Iterable objects
    - We will look at this when we discuss arrays
- See ex6.php

Lecture 3: PHP Arrays

- Arrays in PHP are quite versatile
  - See http://php.net/manual/en/language.types.array.php
- We can use them as we use traditional arrays, indexing on integer values
- We can use them as hashes
  - Associating a key with a value in an arbitrary index of the array
- In either case we access the data via subscripts
  - In the first case the subscript is the integer index
  - In the second case the subscript is the key value
- We can even mix the two if we'd like

- Creating Arrays
- PHP Arrays can be created in a number of ways
  - Explicitly using the array() construct
  - Implicitly by indexing a variable
  - Since PHP has dynamic typing, you cannot identify a variable as an array except by assigning an actual array to it
  - If the variable is already set to a string, indexing will have undesirable results – indexes the string!
  - However, we can unset() it and then index it
  - We can test a variable to see if it is set (isset() and if it is an array (is_array()) among other things
  - Size will increase dynamically as needed

Lecture 3: More on PHP Arrays

- Accessing Arrays – can be done in many ways
- We can use **direct access** to obtain a desired item
    - Good if we are using the array as a hash table or if we need direct access for some other reason
    - We provide the key and retrieve the value
- For **sequential access**, the foreach loop was designed to work with arrays
    - Iterates through the items in two different ways

      foreach ($arrayvar as $key => $value)

      > Gives both the key and value at each iteration

      foreach ($arrayvar as $value)

      > Gives just the next value at each iteration

Lecture 3: PHP Arrays

- How can these both be done efficiently?
  - PHP arrays are not implemented in the traditional way
    - Ex: In Java or C++ the array is a contiguous collection of memory locations
  - PHP arrays more resemble a linked list (see Figure 9.3 in Sebesta text)
    - But wouldn't this not allow direct access?
  - The locations are also hashed
    - The "key" in PHP arrays is actually a hash value
  - So sequential access follows the linked list
  - Direct access accesses via the hash value

Lecture 3: More on PHP Arrays

- Be careful – iteration via foreach is in the order the data has been generated, not by index order
    - i.e. it follows the linked list
    - Thus, even arrays with identical keys and values can have different orderings
- Items accessed in the arrays using foreach are copies of the data, not references to the data
    - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
    - To do this we must change the value using indexing
- A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

Lecture 3: More on PHP Arrays

- The data in the array is not contiguous, so incrementing a counter for the next access will not work correctly unless the array index values are used in the "traditional" way

```
for (int $i = 0; $i < count($A); $i++):
        echo "$A[$i] <br/>";
endfor;
```

- We know that there are count($A) items in $A
- What we do NOT know, is under which indices they are being stored
- There is no requirement that they have to start at 0 or even be integers at all
    - See ex7.php

- In addition to foreach, there are other array iterators that we can use
- Ex: Using next to access the array elements
  - The next() function gives us the next value in the array with each call
    - It moves to the next item, then returns it, so we must get the first item with a separate call (ex: use current())

```
$curr = current($a1);
while ($curr):
    echo "\$curr is $curr <br/>\n";
    $curr = next($a1);
endwhile;
```

Lecture 3: More on PHP Arrays

- Ex: Using each to iterate:
    - The each() function returns a pair with each call
    - A key field for the current key
    - A value field for the current value
    - It returns the next (key,value) pair, then moves, so the first item is no longer a special case

```
while ($curr = each($a1)):
    $k = $curr["key"];
    $v = $curr["value"];
    echo "key is $k and value is $v <BR />\n";
endwhile;
```

- This function may be preferable to next() if it is possible that FALSE or an empty string or 0 could be in the array
    - The loop on the previous slide will stop for any of those values

Lecture 3: More on PHP Arrays

- Both of these iteration functions operate similar to the **Iterator interface** in Java
    - Iterate through the data in the collection without requiring us to know how that data is actually organized
    - However, **unlike in Java**, if the array is changed during the iteration process, the current iteration is NOT invalidated
        - Since new items are always added at the "end" of the array (from an iterator’s point of view) adding a new item during an iteration does not cause any data validity problems
        - However, we need to be careful if doing this – can lead to an infinite iteration

Lecture 3: Sorting PHP Arrays

- There are various predefined sort functions in PHP
- sort (rsort for reverse)
    - Sorts arrays of numbers numerically
    - Sorts arrays of strings alphabetically
    - If mixed, the strings count as 0 compared to numbers
    - Reindexes array so that keys start at 0 and increment from there
- asort
    - Same as sort but retains the original key values (arsort for reverse)

Lecture 3: Sorting PHP Arrays

- PHP uses Quicksort to sort arrays
- This means that PHP sorting is NOT STABLE
- What does it mean for a sort to be STABLE?
  - Given equal keys K_1 and K_2, their relative order before and after the sort will be the same
- Due to data movement during partition, Quicksort is not stable
  - Implications?
  - If we want stability, we will have to do it ourselves
    - See Web for some solutions
- See ex8.php

Lecture 3: Two-dimensional Arrays

- Array values can be any legal PHP type
- This includes the array type, and allows for arbitrary dimensional arrays
- We may think of them as "arrays of arrays"
- It seems odd but once you know the array syntax it follows quite naturally

```
$a[0] = array(1,2,3,4);
$a[1] = array(5,6,7,8);
$a[2] = array(9,10,11,12);
```

Lecture 3: Two-dimensional Arrays

- We can also use "normal" indexing for 2-D PHP arrays
- Keep in mind that the key values are still arbitrary, so we need to be careful
- More general access can be done via iterators or recursive functions – we will see this soon
    - See ex9.php