# Arrays
# and List Data
## Numbers

Podstawy programowania

Jolanta Bachan
2007-11-28

# Arrays and List Data

# Arrays and List Data

- A **list** is an ordered set of scalar data. An **array** is a variable which holds a list. Each *element* of the array is a separate scalar variable with an independent scalar value. These values are ordered – that is, they have a particular sequence from the lowest to the highest element.
- Arrays can have any number of elements. The smallest array has no elements.

# Literal Representation

- A *list literal*:
  - is the way you represent the value of the list within your program;
  - consists of comma-separated values enclosed in parentheses. These values form the elements of the list.

```
(1,2,3) is an array of three values 1, 2, and 3
(“fred”,4.5) – two values, “fred” and 4.5
```

# Literal Representation

- The elements of a list are not necessarily constants

```
($a,17) – two values: the current value of $a,
           and 17
($a+$b,$d+$e) – two values
```

# Literal Representation

- The empty list (with no elements) is represented by an empty pair of parentheses.

```
() – the empty list with zero elements
```

# List constructor function

- It is indicated by two scalar values separated by two consecutive periods.
- This function creates a list of values starting at the left scalar value and continuing up through the right scalar value, incrementing by one at each value.

```
(1..5) – same as (1,2,3,4,5)
(2..6,10,12) – same as (2,3,4,5,6,10,12)
($a..$b) – range determined by the current values
  of $a and $b
```

# Variables

- An array variable holds a single list value (zero or more scalar values).
- Array variable name starts with `@`

`@fred` – is the array variable `@fred`

`@A_Very_Long_Variable_Name`

# Variables

- An array variable holds a single list value (zero or more scalar values).
- Array variable name starts with `@`

`@fred` – is the array variable `@fred`

`@A_Very_Long_Variable_Name`

- The array variable `@fred` is unrelated to the scalar variable `$fred`.

# Array Assignment Operator

```
@fred = (1,2,3) ;
@barney = @fred ;
```

# Array Assignment Operator

```
@fred = (1,2,3) ;
@barney = @fred ;
```

If you assign a scalar value to an array variable, the scalar value becomes the single element of an array:

```
@huh = 1 ; 1 is promoted to the list
 (1) automatically that is, @huh now
 is (1)
```

# Array Assignment Operator

@fred = (3,4,5) ;

@barney = (1,2,@fred,6) ;  @barney becomes (1,2,3,4,5,6)

@barney = (0,@barney) ; – puts 0 in front of @barney

@barney = (@barney,7) ; – puts 7 at the end of @barney

@barney is now (0,1,2,3,4,5,6,7)

# Array Assignment Operator

- Note that the inserted array elements are at the same level as the rest of the literals: a list cannot contain another list as an element.

```
@barney = (1,2,@fred,6) ;
```

# Array Assignment Operator

($a,$b,$c) = (1,2,3) ; - give 1 to $a, 2 to $b, 3 to $c

($a,$b) = ($b,$a) ; - swap $a and $b

($d,@fred) = ($a,$b,$c) ; – give $a to $d and ($b,$c) to @fred

($e,@fred) = @fred; – remove the first element of @fred to $e, this makes @fred = ($c) and $e = $b

# Length of the Array

- To get the length of the array you assign an array variable to a scalar variable

```
@fred = (1,2,3) ;
$a = @fred ; - $a gets 3, the
  current length of @fred
```

# Length of the Array

- To get the length of the array you assign an array variable to a scalar variable

```
@fred = (1,2,3) ;
$a = @fred ; - $a gets 3, the current length of @fred
```

LEARN THIS!!!

# Array Element Access

- The array elements are numbered using sequential integers, beginning at 0, and increasing by 1 for each element. The fisrt element of the `@fred` array is acessed as `$fred[0]`.

# Array Element Access

- The array elements are numbered using sequential integers, beginning at 0, and increasing by 1 for each element. The fisrt element of the `@fred` array is acessed as `$fred[0]`.

?

# Array Element Access

- The array elements are numbered using sequential integers, beginning at 0, and increasing by 1 for each element. The fisrt element of the `@fred` array is acessed as `$fred[0]`.

Array variable

Scalar variable
(part of the array)

# Array Element Access

```
@fred = (7,8,9) ;
$b = $fred[0] ; - give 7 to $b
 (first element of @fred)
$fred[0] = 5 ; now @fred = (5,8,9)
```

# Array Element Access

```
@fred = (5,8,9) ;
$c = $fred[1] ; - give 8 to $c
$fred[2]++ ; - increment the 3rd element of @fred
$fred[1] += 4 ; add 4 to the 2nd element
($fred[0], $fred[1]) = ($fred[1], $fred[0]) ; swap the first two
```

# Array Element Access

@fred = (7,8,9) ;

@fred[0,1] ; – same as ($fred[0],$fred[1])

@fred[0,1] = @fred[1,0] ; – swap the first two elements

@fred[0,1,2] = @fred[1,1,1] ; – make all 3 elements like the 2nd

@fred[1,2] = (9,10) ; - change the last two values to 9 and 10

# Array Element Access

```
@fred = (7,8,9) ;
$a = 2 ;
$b = $fred[$a]; – like $fred[2] or 9
$c = $fred[$a-1]; - $c gets
 $fred[1], or 8
($c) = (7,8,9)[$a-1] ; ($c) gets 8
```

# Array Element Access

```
@fred = (7,8,9) ;
@barney = (2,1,0) ;
@backfred = @fred[@barney] ; # same
  as @fred[2,1,0], or ($fred[2],
  $fred[1], $fred[0])
```

# Array Element Access

`$#fred` is used to get the index value of the last element of `@fred`.

```
@fred = (7,8,9) ;
$last_subscript = $#fred ;
print "This is the last subscript:
  $last_subscript" ;
```

# Array Element Access

A negative subscript on an array counts back from the end. So, another way to get the last element is with the subscript `-1`. The second to the last element would be `-2`, and so on.

```
@fred = (7,8,9) ;
print $fred[-1] ; # prints 9
print $#fred ; # prints 2
print $fred[$#fred] ; # prints 9
```

# push and pop Functions

- An array is commonly used as a stack of information, where new values are added to and removed from the right-hand side of the list.

# push and pop Functions

```
@mylist = (1,2,3) ;
$new_value = 4 ;
push(@mylist, $new_value) ;
 # like @mylist = (@mylist,
 $new_value)
$old_value = pop(@mylist) ; #
 removes the last element of @mylist
push (@mylist, 4,5,6) ; # @mylist =
 (1,2,3,4,5,6)
```

# `shift` and `unshift` Functions

- `shift` and `unshift` functions perform actions of the “left” side of a list (the portion with the lowest subscript).

# `shift` and `unshift` Functions

- `shift` and `unshift` functions perform actions of the “left” side of a list (the portion with the lowest subscript).

```
unshift(@fred,$a) ;
 # like @fred = ($a,@fred)
unshift(@fred,$a,$b,$c) ;
   # like ($a,$b,$c,@fred)
$x = shift(@fred) ; # like
 ($x,@fred) = @fred
```

# shift and unshift Functions

- shift and unshift functions perform actions of the “left” side of a list (the portion with the lowest subscript).

```
@fred = (5,6,7) ;
unshift(@fred,2,3,4) ; # @fred is
  now (2,3,4,5,6,7)
$x = shift(@fred) ; # $x gets 2,
  @fred is now (3,4,5,6,7)
```

# The `reverse` Function

- The `reverse` function reverses the order of the elements of its arguments, returning the resulting list.

```
@a = (7,8,9) ;
@b = reverse(@a) ; # gives @b the value of (9,8,7)
@b = reverse(7,8,9) ; # same thing
```

# The `reverse` Function

- Note that the argument list is unaltered; the `reverse()` function works on a copy. To reverse an array “in place”, you'll need to assign it back into the same variable.

```
@b = reverse(@b) ; # give @b the
  reverse of itself
```

# The `sort` Function

- The `sort` function takes its arguments, and sorts them as if they were single strings in ascending ASCII order. It returns the sorted list, without alternating the original list.

```
@x = (1,2,3,6,14,25,30) ;
@y = sort(@x) ; # @y gets
  (1,14,2,25,3,30,6)
```

See you next week!