Arrays

Arrays – 1/3

- An array holds a lot of scalar variables
- **Starts with @**
- Any data type can be part of list – scalars, other arrays, reference variables, numeric literals and string literals
- Each array can hold as many scalar variables as the machine's memory and disk space

Arrays – 2/3

- **Each scalar variable is kept in separate cell**
- **Each array cell has unique identifier**, called **index**, to reference the scalar data stored
- **An array's scalar values are stored sequentially**

Arrays – 3/3

```
@name= (“Steven”, “Hermann”);
print $name[0];
print $name[1];
```

- Result

  StevenHermann

Array cell assignment

- To assign any type of data into array, we need to **identify the cell into which we want to store data**

Example

```
$index = 0;
$mixdata[$index] = \@mixdata;
$mixdata[2] = 727;
$mixdata[5] = “last cell”;
$mixdata[3] = 3.44;
foreach $value (@mixdata)
{
      print “cell no $index = $value \n”;
                  $index++;
}
print “\n last cell of array mixdata is
$#mixdata\n”;
```

- Result

  cell no 0 = ARRAY<0x11f6d8c>

  cell no 1 =

  cell no 2 = 727

  cell no 3 = 3.44

  cell no 5 = last cell

  last cell of array mixdata is 5

- **Indexes to array must be numeric**
- **Can also use scalar variables that contain numeric values**
- Can also index array using **range operator (..)**
- **Indexing array using range operator is called slicing**

Array cell assignment -Rules – 1/2

$array[cellindex] = scalar

- Value assigned to array cell must be scalar or variable that resolves to scalar
- Cell index must be scalar or variable that resolves to scalar
- Square brackets are required to identify all array assignments

Array cell assignment - Rules – 2/2

- **$ takes place of @ when performing scalar assignment**
- Assignment to array cells that skip over array cells that have not been assigned **initializes all skipped array cells to null**
- **Arrays grow in size as new cells are added**

Array list assignment – 1/5

- Lists like arrays can be of **mixed types**
- String literals, numeric literals, scalar variables, arrays, array slices and references are all valid pieces of single list that can be assigned to array

Array list assignment – 2/5

```
@mixdata = (\@mixdata,,28,,3.22,,"last
      cell");
printArray(@mixdata);
@mixdata = (\@mixdata, ' ' ,28, ' ',3.22,
                  ' ' ,"last cell");
printArray(@mixdata);
@pieces = @mixdata[4..8];
printArray(@pieces);
@all = @mixdata;
printArray(@all);
```

Array list assignment – 3/5

```
sub printArray
{
  my @localArray = @_;
  $index = 0;
  foreach $value (@localArray)
  {
       print "cell no $index = $value\n";
              $index++;
  }
  print "last cell of array is $#localArray\n';
}
```

Array list assignment – 4/5

- **Result**

cell no 0 = ARRAY<0x11f6d68>
cell no 1 = 28
cell no 2 = 3.22
cell no 3 = last cell
last cell of array is 3
cell no 0 = ARRAY<0x11f6d68>
cell no 1 =
cell no 2 = 28
cell no 3 =
cell no 4 = 3.22
cell no 5 =
cell no 6 = last cell
last cell of array is 6

Array list assignment – 5/5

cell no 0 = 3.22
cell no 1 =
cell no 2 = last cell
cell no 3 =
cell no 4 =
last cell of array is 4
cell no 0 = ARRAY<0x11f6d68>
cell no 1 =
cell no 2 = 28
cell no 3 =
cell no 4 = 3.22
cell no 5 =
cell no 6 = last cell
last cell of array is 6

Array list assignment – Rules – 1/2

- Assign a list to array

  @array = (1, “string”, 3.5, \@array2);

- Assign array to array

  @array1 = @array2;

- Assign array slice

  @array1 = @array2[1..4];

Array list assignment – Rules – 2/2

- Array **grow in size** to accommodate data
- Assignments made to existing arrays **overwrite** original array

```
@array1 = (1..10);
@array2 = (11..20);
@array1 = @array2;
```

Array sizing

- **End of array** is defined by special variable **$#arrayname**, where arrayname is actual name

  $#mixdata, $#name

- This **returns the index of last cell of named array**

- **$#arrayname is referred as “last cell index variable”**

Setting array size

- When we know the actual size of the array we are going to create, we can initialize the size of the array by setting the $#arrayname variable
- Suppose we have 2000 items in catalog. Our price list contains 2000 cells
- Initialize price list

  $#pricelist = 2_000;

Adding cells to array

- **Can use the last cell index variable to add cells to end of array**
- **Increment the last cell index variable before assigning new value**
- So array will grow in size and accept new value

  $arrayname[++$#arrayname] = $var;

  OR

  $arrayname[$#arrayname+1] = $var;

Deleting cells from array – 1/2

- Can use last cell index variable to shrink size of array
- Can also delete cells by decrementing last cell index
  $#arrayname--;
- Suppose we have 10 digit array named @digits
  @digits = (0..9);
- And then decrement it
  --$#digits;

Deleting cells from array – 2/2

- When we access $digits[9], we get **null**
- **Each deleted cell is assigned null value**
- **To reinitialize the array to all null values, set the last cell index to -1**

```
$#digits = -1;
```

- Then array will be **empty**

Changing first cell index

- Special variable **$[** defines first cell index of array cell
- **Variable defaults to zero**, but can be set to any number
- **Do not recommend to change this index**

Calculating array size

- Calculate size of array using last cell index variable

  $#arrayname - $[+ 1;

- Easier way

  $arraysize = @arrayname;

- $arraysize will be assigned no of elements in the array @arrayname counting from 1

Array data retrieval – Rules – 1/4

- Retrieve single cell of array in scalar context

  $scalar = $array[index];

- To retrieve all or part of array, use list context

  @array = @array2[2..4];

Array data retrieval – Rules – 2/4

- Copy complete array, a contiguous slice of cells, or random list of cells by referencing array in list context

```
#copy entire array
@array1 = @array2;
# copy array slice
@array1 = @array2[3..5];
#copy pieces of one array
@array1 = @array2[0,2,5..8,4,10,..12];
```

Array data retrieval – Rules – 3/4

- Process array in list context

```
foreach $item (@arrayname)
{
}
```

- Index from front to last of array

```
for($index=0;$index<=$#arrayname;$index++)
  {
    $item = $arrayname[$index];
  }
```

Array data retrieval – Rules – 4/4

- Index from end of array to front

```
for ($index = -1;
  abs($index)<= $#arrayname - $[ + 1;
  $index--)

{
  $item = $arrayname[$index];
}
```

Array slices – Rules – 1/6

- Array slices are always accessed in list context
  @arrayname[scalarlist];
- An array slice can be used as lvalue
  @arrayname[@digits] = (11..20);
- Array reference does not need to reference consecutive elements of array
  @oddno = @digits[1,3,5];
- Array slice can reference elements out of order
  @random = @digits[2,0,3,10];

Array slices – Rules – 2/6

```
@digits = (11..21);
@slice[10..20] = (@digits);
print “contents of array : @slice\n”;
print “last index is $#slice\n”;
@09 = (0..9);
@slice[@09] = (@digits);
print “contents of array : @slice\n”;
print “last index is $#slice\n”;
@slice[1,3,5,7,9] = (2,4,6,8,10);
@even = @slice[1,3,5,7,9];
print “contents of array : @even\n”;
print “last index is $#even\n”;
```

Array slices – Rules – 3/6

```
  @slice[@digits,77,55,33] = (222, 23, 44, 55,
  33, 10, 22, 221, 11, 44, 22, 1, 33, 12, 34);

print "contents of array : @slice\n";

print "indexes 55, 33, 77 and 12 are :
  @slice[55, 33, 77, 12] \n";

print "last index is $#slice\n";
@names = (tom, john, james, jack, jill);

printNames(@names);
```

Array slices – Rules – 4/6

```
sub printNames(@)
{
        my (@names)= @_;
        for ($i=0; $i<=$#names;)
        {
                print "$names[$i++]\t";
        }
}
```

Array slices – Rules – 5/6

- **Result**

  contents of array : 11 12 13 14 15 16 17 18 19 20 21

  last index is 20

  contents of array : 11 12 13 14 15 16 17 18 19 20 11 12 13 14 15 16 17 18 19 20 21

  last index is 20

  contents of array : 2 4 6 8 10

  last index is 4

  contents of array : 11 12 13 14 15 16 17 18 19 20 11 12 13 14 15 16 17 18 19 20 21 222, 23, 44, 55, 33, 10, 22, 221, 11, 44, 22, 1, 33, 12, 34

  indexes 55, 33, 12 are 33 12 23

  last index is 77

  tom john james jack jill

Array slices – Rules – 6/6

- When we use array inside array index brackets, array resolves into list of scalars

@slice[@09] = (@digits);

Built in functions- 1/12

Push

- **Adds elements** to **end of array**

push @array, element

- Element may be scalar variable or array

```
@digits09 = (0..9);
push @array, @digits09;
```

Built in functions- 2/12

Unshift

- Put elements into **first cell index** of array instead of last cell index
- **It inserts at front of array**

```
@digits02 = (0..2);
unshift @digits02, 3;
@digits = (10..15);
unshift @digits, 3, 4, 5, 6);
```

Built in functions- 3/12

- **Pop**
  - **Remove last element from array**
  - If array is **empty**, pop returns **undefined**

```
@array = (1..10);
$var = pop @array;
```

Built in functions- 4/12

Shift

- **Removes the first element from array**

```
@digits02 = (0..2);
$digit = shift @digits02;
```

Built in functions- 5/12

- **Splice**
  - **Modify or delete elements from array**
  - Takes a **starting index**, **no of cells to modify**, and **list of items to modify** in those cell locations
  - If we **omit list of elements to modify**, splice **removes no of elements we specify**

Built in functions- 6/12

- If **no of elements to remove is not given**, it **removes all of the elements to end of array**, **beginning with cell index** we gave

```
@digits = (0..9);
splice @digits, 5, 1;
```

@digits become 0,1,2,3,4,6,7,8,9

Built in functions- 7/12

- Remove all the elements after and including 5th element

```
@digits = (0..9);
splice @digits, 5;
```

@digits contain 0,1,2,3,4

Built in functions- 8/12

- To modify 5th element in array

```
@digits = (0..4, 6..9);
splice @digits, 5, 1, 5;
```

@digits contain 0,1,2,3,4,**5**,7,8,9. The **no 6 is now 5**

- **Splice does not insert new elements to array**

Built in functions- 9/12

- **Sort**
  - **Sort on character strings**

```
@names = (Eric, Tom, James, Pete);
@names = sort @names;
```

Built in functions- 10/12

- Problem with numbers

```
@nos = (12,34,22,55,11,44);
@nos = sort @nos;
print "@nos\n";
@nos = sort sortnos @nos;
print "@nos\n";
sub sortnos
{
      $a <=> $b;
}
```

Built in functions- 11/12

- Result

12 11 22 34 44 55
11 12 22 34 44 55

Built in functions- 12/12

- **Reverse**
  - **Reverse the array**

```
@names = (Eric, Tom, James, Pete);
@names = reverse @names;
print “@names\n”;
```

  - Result

    etePsemaJmoTcirE