Examples of Using the ARGV Array

# mimics the Unix echo utility
foreach (@ARGV) {
    print "$_ ";
}
print "\n";

# count the number of command line arguments
$i = 0;
foreach (@ARGV) {
    $i++;
}
print "The number of arguments is $i.\n";

Loop Control Operators

• Perl has three loop control operators.
  – last: used to break out of a loop
  – next: used to goto the next iteration
  – redo: used to repeat the current iteration

Last Operator

• The *last* operator breaks out of the innermost loop in which it is contained. This is similar to
  the *break* statement in C.

# Sums the first 100 numbers read or
# entire input if less.
$i = 1;
$sum = 0;
while ($num = <STDIN>) {
    chomp($num);
    $sum += $num;
    if ($i++ == 100) {
        last;
    }
}

Next Operator

• The *next* operator skips over the rest of the loop body and continues with the next iteration. This
  operator is similar to the *continue* statement in C.

# sums the positive elements of the
# array vals
$sum = 0;
foreach $val (@vals) {
    if ($val <= 0) {
        next;
    }
    $sum += $val;
}
**Redo Operator**

- The *redo* operator will go back to the top of the loop block, but without performing the increment portion, testing the loop condition, or advancing to the next value in the list.

```perl
foreach $s (@strings) {
    print “Do you wish to print $s?
    "; my chomp($ans = <STDIN>);
    if ($ans eq “yes”) {
        print $s, “
    ”;
    } elsif ($ans ne “no”) {
        print “‘$ans’ is not a valid answer.
    ”;
    redo;
}
}
```

**Reverse Operator**

- The *reverse* operator takes a list or array of values as input and creates a new list with the values in reverse order.

```perl
@nums = 1..100;   # @nums = (1, 2, ..., 100);
@revnums = reverse @nums;
    # @revnums = (100, 99, ..., 1);
@revnums = reverse 1..100;
    # @revnums = (100, 99, ..., 1);
@nums = reverse @nums;
    # reverses @nums itself
```

**Reverse Operator in Scalar Context**

- The *reverse* operator can be used in either an array or scalar context. In a scalar context it returns a reversed string after concatenating all of the strings in the list.

```perl
@animals = qw/ dog cat cow /;
@backwards = reverse @animals;    # (“cow”, “cat”, “dog”)
$backwards = reverse @animals;    # “woctacgod”
$backone = reverse ($animals[1]);  # “tac”
@nums = (1, 9, 23);
$s = reverse @nums;               # ?
```

**Sort Operator**

- The *sort* operator takes a list or array of values as input and creates a sorted list in ASCII order.

```perl
@fruit = qw( apple orange grape pear lemon );
@sortedfruit = sort @fruit;    # (apple grape lemon orange pear)
print “@sortedfruit
”;
foreach $f (sort @fruit) {
    print $f, “
”; # prints fruit in sorted order
}
@nums = sort 98..101;   # assigns (100, 101, 98, 99)
$n = sort 98..101;     # assigns undef
```
Hashes

- A hash is similar to an array in that individual elements are accessed by an index value and may have an arbitrary number of values.
- A hash differs from an array in that the indices are strings, which are called keys.
- The elements of a hash have no particular order.
- The hash contains key-value pairs. The keys have to be unique, but the values may not.
- A hash can be viewed as a very simple database, where a scalar data value can be filed for each key.

Why Use a Hash?

- There are often relationships between sets of data that need to be maintained. You would like to efficiently access one set of data by using the key from another.
- Examples
  - word => meaning
  - student ID => name
  - loginname => name
  - employee ID => salary
  - title => author
  - barcode => price

Hash Declarations

- Use the '%' preceding a name to identify a hash.
  ```perl
  my %book;
  my %products;
  ```
- The names of hashes are kept in a separate namespace from scalars and arrays. However, it is good practice to use a unique name for each hash.

Hash Element Access

- General form. Use '$' before the hashname to access an individual scalar value from a hash. Use '{' '}' instead of '[' ']' so that Perl will know it is a hash element instead of an array element being accessed.
  ```perl
  $hashname{$keyvalue}
  ```
- If the $keyvalue contains a number or an expression, then the value is converted to a string, which is input to the hash function.
Hash Element Access Examples

```perl
$names{67415} = "Doe, John";  # storing a name
$names{67415} = "Doe, Jane";  # name overwritten
$name = $names{67415};        # retrieving a name
$name = $names{46312};        # invalid key returns an undef value
$names{$id} = "Smith, Tom";   # storing another name
foreach $id (@student_ids) {  # for each id
    print "$id=$names{$id}\n";  # print id=name
}
```

Referring to the Entire Hash

- Use the '%' character to refer to the entire hash.
```perl
%new_hash = %old_hash;       # copy an entire hash

# initialize a hash by specifying key-value pairs
%fruit = ("apple", 0.30, "orange", 0.45, "pear", 0.50);

# can use '=>' instead of a ','
%fruit = ("apple" => 0.30, "plum" => 0.45, "pear" => 0.50);

# cannot print an entire hash directly
print "%fruit\n";              # prints "%fruit"

# can turn a hash back into an array of key-value pairs
@fruitarray = %fruit;
```

Keys and Values Function

- The `keys` function takes a hashmap name as input and creates a list of the current keys in the hash.
- The `values` functions takes a hashmap name as input and creates a list of the current values in the hash.
```perl
# hash initialization
%fruit = ("apple" => 0.30, "plum" => 0.45, "pear" => 0.50);

@k = keys %fruit;  # "apple", "plum", "pear" in some order
@v = values %fruit; # 0.30, 0.45, 0.50 in some order
```

Each Function

- The `each` function takes a hash name as input and returns a two element list (key-value pair) for each iteration of a loop.
```perl
# print the name and price of each type of fruit
while ( ($name, $price) = each %fruit) {
    print "$name = $price\n";
}
```
Exists Function

- The exists function checks if a key exists in a hash. Note this function returns a true or false value, not the value associated with the key.

```perl
if (exists $fruit{$f}) {
    print "The price of an orange is $fruit{$f}.
";
}
```

Delete Function

- The `delete` function removes a key-value pair from a hash.

```perl
# hash initialization
%fruit = ("apple" => 0.30, "plum" => 0.45, "pear" => 0.50);
delete $fruit{"plum"};  # deletes "plum" => 0.45
@fruitarray = %fruit;   # assign to an array
print "@fruit
";         # only two key-value pairs will be printed
```

Formatted Output with Printf

- The Perl `printf` function, unlike the `print` function, takes a format string as its first argument. Typically only used to print scalars.
- The format string has similar conversions as the C `printf` function.
- This feature should be used when you want more control over how the output should appear.

- %s: string
- %d: truncated decimal
- %f: float

Example Printf's

```perl
printf "%7d
", $i;  # Prints integer value of $i right justified in 7 columns on one line.
printf "%-10s", $s;  # Prints $s as a left justified string in 10 columns.
printf "%6.2f", $f;  # Prints $f in a 6 column field with 2 digits after the decimal point (ddd.dd).
printf "%${max}s", $s;  # Prints $s as a right justified string in a field $max columns wide. # Note the use of the {}.
printf "%s=%d\n", $name, $val;  # Prints $name as a string, followed by an '=' followed by $val as an integer.
```