```
#!/usr/local/bin/perl -w

# <STDIN> IN LIST CONTEXT
# By Louis Ziantz

# Description: This program is a demonstration of reading
#               from STDIN into an array.
# Procedure:   Several lines are read from input and
#             the array is output several ways.
# Input:       Lines of text from STDIN.
# Output:      Array read from STDIN in several ways.

print("Type several lines followed by a <control>D ...\n");
@a = <STDIN>;
# @a is ("<STDIN> in list context.\n",
#        "Each element of @a gets a line of input.\n",
#        "Like so.\n")
print("Result of print(\@a); ... \n");
print(@a);
print("Result of print(\"\@a\"); ... \n");
# an easier way to print the same message
# print('Result of print("@a"); ...', "\n");
print("@a");
```

```
# Output:
% array.stdin.plx
Type several lines followed by a <control>D ...
<STDIN> in list context.
Each element of @a gets a line of input.
Like so.
Result of print(@a); ...
<STDIN> in list context.
Each element of @a gets a line of input.
Like so.
Result of print("@a"); ...
<STDIN> in list context.
 Each element of @a gets a line of input.
 Like so.
# notice that spaces are added when we
# interploate the array, leads to ...
```

Comparison Operators

Note : Return value assumes $l <op> $r is evaluated

Comparison	Numeric	String	Return value
Equal	==	eq	True if $l is equal to $r
Not Equal	!=	ne	True if $l is not equal to $r
Less than	<	lt	True if $l is less than $r
Greater than	>	gt	True if $l is greater than $r
Less than or equal	<=	le	True if $l is not greater than $r
Gr. than or equal	>=	ge	True if $l is not less than $r
Comparison	<=>	cmp	0 if equal, 1 if $l greater, -1 if $r greater

Logical Operators

Note : Result value assumes $l <op> $r is evaluated
(or <op> $a for unary operator)

Name	High-prec	Low-prec	Result
And	&&	and	$l if $l is false; $r otherwise
Or	||	or	$l if $l is true; $r otherwise
Not	!	not	True if $a is not true

Assoc	Operator
Right	** (2 ** 2 ** 3 # 2 ** (2 ** 3))
Right	! + - (unary)
Left	* / % x (multiplication)
Left	+ - . (addition)
Nonassoc	Named unary ops (like chomp)
Nonassoc	< <= > >= lt le gt ge
Nonassoc	== != <=> eq ne cmp
Left	&&
Left	||
Nonassoc	.. (range)
Right	= += -= (etc.) (assignment)
Right	not
Left	and
Left	or

```
#!/usr/local/bin/perl -w

# NESTED IF IN PERL EXAMPLE
# By Louis Ziantz

# Description: This program is a demonstration a nested
#               if structure in Perl.
# Procedure:   A grade is read, and an appropriate
#             grade counter is incremented based on
#             the grade.
# Input:       A letter grade.
# Output:      None.

$a_count = $b_count = 0;
$c_count = $d_count = 0;
$f_count = 0;

$grade = <STDIN>;
chomp($grade);
if (($grade eq 'a') || ($grade eq 'A')) {
    $a_count++;
} elsif (($grade eq 'b') || ($grade eq 'B')) {
    $b_count++;
} elsif (($grade eq 'c') || ($grade eq 'C')) {
    $c_count++;
} elsif (($grade eq 'd') || ($grade eq 'F')) {
    $d_count++;
} elsif (($grade eq 'f') || ($grade eq 'F')) {
    $f_count++;
} else {
    print("Invalid grade $grade entered\n.");
}
```

```
$grade = <STDIN>;
chomp($grade);
if ("\U$grade" eq 'A') {
    $a_count++;
} elsif ("\U$grade" eq 'B') {
    $b_count++;
} elsif ("\U$grade" eq 'C') {
    $c_count++;
} elsif ("\U$grade" eq 'D') {
    $d_count++;
} elsif ("\U$grade" eq 'F') {
    $f_count++;
} else {
    print("Invalid grade $grade entered\n.");
}
```

```
#!/usr/local/bin/perl -w

# VARIOUS HASH EXAMPLES IN PERL
# By Louis Ziantz

# Description: This program is a demonstration of assigning
#              to hash values, building hashes from lists,
#              reversing hashes, and iterating through hashes.
# Procedure:   Omitted to save space.
# Input:       None.
# Output:      A hash is printed using two different methods.

$a{"aaa"} = "bbb"; # creates element "bbb" with key "aaa"
$a{ccc} = "bBB"; # creates element "bbb" with key "ccc"
$a{345.2} = 456.5; # creates element 456.5 with key "345.2"


%building = (
     "CS",   "Bahamas",
     "CS",   "Amos Eaton",
     "Math", "Amos Eaton",
     "ECSE", "CII",
     "Arch", "Greene"
     );


%building = (
     CS   => "Amos Eaton",
     Math => "Amos Eaton",
     ECSE => "CII",
     Arch => "Greene"
    );
```

```
@building = %building;
# ("ECSE", "CII", "Math", "Amos Eaton", "Arch", "Greene",
#  "CS", "Amos Eaton")
@dept = reverse(@building);
# ("Amos Eaton", "CS", "Greene", "Arch", "Amos Eaton",
#  "Math", "CII", "ECSE")
%dept = @dept;
@dept = %dept;
# ("CII", "ECSE", "Greene", "Arch", "Amos Eaton", "Math")
```

```
foreach $key (keys (%building)) { # for each key of %building
    print("Key $key contains $building{$key}.\n");
}
```

```
while (($key,$value) = each(%building)) { # as above
    print("Key $key contains $value.\n");
}
```