Elements of Programming in Perl

<H16-6/7>

ARRAYS
Vectors and Matrices

Josep F. Abril

jabril@imim.es

@rrays in Perl

@ ⇨ **@rrays of scalars**

```
                            $nucleotides[1]
                                  @nucleotides[1,2]

@nucleotides = ( "A",  "C",  "T",  "G" );
                  0     1     2     3
                                     $#nucleotides
```

$#nucleotides + 1 == **scalar** @nucleotides

\ ⇨ **References**

```
                              (@$ary)[1] eq $nucleotides[1]
                              ${$ary}[1] eq $nucleotides[1]
                               $ary->[1] eq $nucleotides[1]
  $ary = \@nucleotides;
  $anon_ary = [];
```

Working with @rrays

```
@nucleotides = ( "A", "C", "T", "G" );
```

push array, list
```
push @nucleotides, "N", "."; #-> ("A","C","T","G","N",".")
```
pop array
```
$nuc = pop @nucleotides;      #-> ("A","C","T","G","N") && $nuc eq "."
```
shift array
```
$nuc = shift @nucleotides;    #-> ("C","T","G","N") && $nuc eq "A"
```
unshift array, list
```
unshift @nucleotides, $nuc;   #-> ("A","C","T","G","N")
```
splice array, offset, length, list
```
splice @nucleotides, 2, 1, $nuc; #-> ("A","C","A","G","N")
@tmp = splice @nucleotides, 2, 2; #-> ("A","C","N") && @tmp = ("A","G")
```
reverse list
```
@tmp = reverse @nucleotides;  #-> ("N","C","A")
```
join string, list
```
print STDOUT join(":",@nucleotides),"\n"; #-> prints "N:C:A"
```

Sorting Array Elements

```
@nuc = qw / G A T C /;    @pos = ( 5, 20, 10, 1 );
```

sort block list

```
@sorted = sort @nuc;  #-> @sorted = ("A","C","G","T")
@sorted = sort @pos;  #-> @sorted = ("1","10","20","5") ???
```

`sort` works by default on **string** context not on **numeric**

Forcing **string** context:

```
@sorted = sort { $a cmp $b } @nuc;  #-> @sorted = ("A","C","G","T")
@sorted = sort { $a cmp $b } @pos;  #-> @sorted = ("1","10","20","5")
```

Forcing **numeric** context:

```
@sorted = sort { $a <=> $b } @nuc;  #-> @sorted = ("G","A","T","C")
@sorted = sort { $a <=> $b } @pos;  #-> @sorted = ("1","5","10","20")
```

Reverse sorting:

```
@sorted = reverse sort { $a cmp $b } @nuc;

@sorted = sort { $b cmp $a } @nuc;  #-> @sorted = ("T","G","C","A")
@sorted = sort { $b <=> $a } @pos;  #-> @sorted = ("20","10","5","1")
```

Building Matrices

```
@matrix = ( [ 1, 2, 3 ],    # Taking advantage
            [ 4, 5, 6 ],    # of anonymous arrays
            [ 7, 8, 9 ] );  # to build N-dimensional matrices...

print "@matrix\n"; #-> "ARRAY(0x812c7a0) ARRAY(0x812c6bc) ARRAY(0x813ab44)"
print scalar @matrix,"\n"; #-> "3"
print "@{ $matrix[1] }\n"; #-> "4 5 6"
print scalar @{ $matrix[0] },"\n"; #-> "3"
print "${ $matrix[0] }[1]\n"; #-> "2"
print STDOUT (@{ $matrix[2] })[1],"\n"; #-> "8"
print "$matrix[1][1]\n"; #-> "5"

# Traversing the matrix
for ($i = 0; $i < scalar(@matrix); $i++) {
    for ($j = 0; $j < scalar(@{$matrix[0]}); $j++) {
        print $matrix[$i][$j]," "; # prints:
    } # for $j                    #         "1 2 3 \n"
    print "\n";                   #         "4 5 6 \n"
} # for $i                        #         "7 8 9 \n"
```

Analyzing Sequence Content (II)

```
#!/usr/bin/perl
use strict;
use warnings;

# initializing variables
my $dna_seq = "ATGCATTGGGGAACCCTGTGCGGATTCTTGTGGCTTTGGCCCTATCTTTTCTATGTCCAAGCTG".
              "TGCCCATCCAAAAAGTCCAAGATGACACCAAAACCCTCATCAAGACAATTGTCACCAGGATCAA";
my %NUCLEOTIDES = ();
```

```
# Looping throught the sequence STRING

my $seqlen = length($dna_seq);

for (my $i = 0; $i < $seqlen; $i++) {

    my $char = substr($dna_seq,$i,1);

    $NUCLEOTIDES{$char}++;

};
```

```
# Looping throught the sequence ARRAY

my @sequence = split //, $dna_seq;

my $seqlen = scalar(@sequence);

for (my $i = 0; $i < $seqlen; $i++) {

    $NUCLEOTIDES{$sequence[$i]}++;

};
```

```
# Printing results
foreach my $nucleotide (sort keys %NUCLEOTIDES) {
    print STDOUT "Total $nucleotide = $NUCLEOTIDES{$nucleotide}\n";
};
```

Two Helpful Modules

```
use Data::Dumper;
```

Stringified perl data structures, suitable for both printing and eval

```
perl -e '
   use Data::Dumper;
   @matrix = ( [ 1, 2 ], [ 3, 4 ] );
   print Data::Dumper->Dump(
                  [ \@matrix ],
               [ qw/ *matrix / ]);
   '
@matrix = [
            [
              1,
              2
            ],
            [
              3,
              4
            ]
          ];
```

```
use Benchmark;
```

Evaluate running times of Perl code

```
perl -e '
   use Benchmark;
   $t0 = new Benchmark;
   # ... code to test starts here ...
   sleep(10);
       # do nothing for 10 seconds...
   # ... code to test  ends  here ...
   $t1 = new Benchmark;
   $td = timediff($t1, $t0);
   print "The code took: \n",
          timestr($td), "\n";
   '
The code took:
10 wallclock secs (0.0 usr + 0.0 sys = 0.0 CPU)
```

Analyzing Sequence Content (III)

```
#!/usr/bin/perl
use strict;
use warnings;
use Data::Dumper;

# initializing variables
my $dna_seq = "ATGCATTGGGGAACCCTGTGCGGATTCTTGTGGCTTTGGCCCTATCTTTTCTATGTCCAAGCTG".
              "TGCCCATCCAAAAAGTCCAAGATGACACCAAAACCCTCATCAAGACAATTGTCACCAGGATCAA";
my %NUCLEOTIDES = ();
```

```
# Looping throught the sequence STRING
my $seqlen = length($dna_seq);
for (my $i = 0; $i < $seqlen; $i++) {
    my $char = substr($dna_seq,$i,1);
    $NUCLEOTIDES{$char}++;
};
```

```
# Looping throught the sequence ARRAY
my @sequence = split //, $dna_seq;
my $seqlen = scalar(@sequence);
for (my $i = 0; $i < $seqlen; $i++) {
    $NUCLEOTIDES{$sequence[$i]}++;
};
```

```
# Printing results
foreach my $nucleotide (sort keys %NUCLEOTIDES) {
    print STDOUT "Total $nucleotide = $NUCLEOTIDES{$nucleotide}\n";
};

# Printing data structures (dumping contents of variables)
print Data::Dumper->Dump([    \$dna_seq, \%NUCLEOTIDES, \@sequence   ],
                         [ qw/ *dna_seq   *NUCLEOTIDES   *sequence / ]);
```

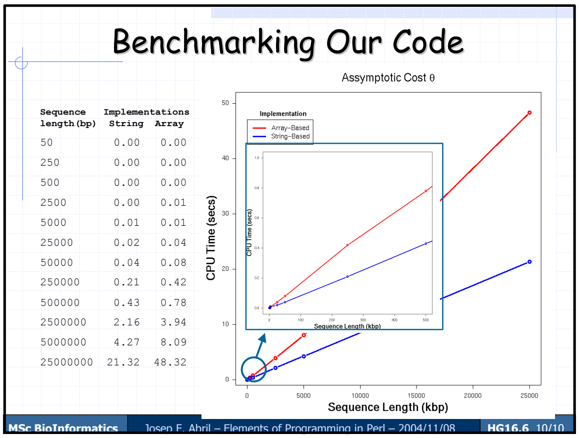
Analyzing Sequence Content (IV)

```
#!/usr/bin/perl
use strict;
use warnings;
use Benchmark;
# initializing vars
my $dna_seq = "ATGCATTGGGGAACCCTGTGCGGATTCTTGTGGCTTTGGCCCTATCTTTTCTATGTCCAAGCTG".
              "TGCCCATCCAAAAAGTCCAAGATGACACCAAAACCCTCATCAAGACAATTGTCACCAGGATCAA";
my %NUCLEOTIDES = ();

$dna_seq x= 10000;

my @exectime = (new Benchmark);
```

```
# Looping throught the sequence STRING
my $seqlen = length($dna_seq);
for (my $i = 0; $i < $seqlen; $i++) {
    my $char = substr($dna_seq,$i,1);
    $NUCLEOTIDES{$char}++;
};
```

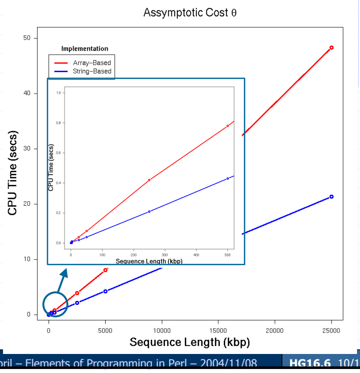
```
# Looping throught the sequence ARRAY
my @sequence = split //, $dna_seq;
my $seqlen = scalar(@sequence);
for (my $i = 0; $i < $seqlen; $i++) {
    $NUCLEOTIDES{$sequence[$i]}++;
};
```

```
# timing previous commands execution
push @exectime, (new Benchmark);
print timestr(timediff($exectime[$#exectime],$exectime[($#exectime - 1)])),"\n";

# Printing results
foreach my $nucleotide (sort keys %NUCLEOTIDES) {
    print STDOUT "Total $nucleotide = $NUCLEOTIDES{$nucleotide}\n";
};
```



Benchmarking Our Code

Sequence length (bp)	Implementations String	Array
50	0.00	0.00
250	0.00	0.00
500	0.00	0.00
2500	0.00	0.01
5000	0.01	0.01
25000	0.02	0.04
50000	0.04	0.08
250000	0.21	0.42
500000	0.43	0.78
2500000	2.16	3.94
5000000	4.27	8.09
25000000	21.32	48.32


Assymptotic Cost θ
Implementation
Array–Based
String–Based
CPU Time (secs)
Sequence Length (kbp)
Sequence Length (kbp)
CPU Time (secs)