Perl for Biologists

Session 8

April 22, 2015

Practical examples

Jaroslaw Pillardy

Review of Session 7

Regular expression: a specific pattern that is used to match strings of text

```
if ($string =~ /Hello/)
if ($string =~ /$match/)
```

Metacharacters and quantifiers:

/./	matches all but newline
/a|b/	matches a OR b
/a*/	zero or more a's
/a+/	one or more a's
/a?/	zero or one a
/a{3}/	exactly 3 repetitions of a

Review of Session 7

Character classes, shortcuts, anchors

/[aeiouy]/	lowercase vowels
/[012345]/	first five numbers, same as /[0-5]/
/[^0-5]/	anything except first five numbers
/\d/	Digit, /[0-9]/
/\D/	Nondigit, /[^0-9]/
/\s/	Whitespace, /[\t\n\r\f]/
/\S/	Nonwhitespace, /[^ \t\n\r\f]/
/\w/	Word character, /[a-zA-Z0-9_]/
/\W/	Nonword character, /[^a-zA-Z0-9_]/
/\A\w+/ or /^\w+/	force matching from start of string
/\w+\z/ or /\w+\Z/ or /w+$/	force matching from the end of string

Review of Session 7

Grouping

( )   groups characters into one unit, saves match results in $1, $2, $3, … for use outside of regex

/(iss){2}/	looks for two repetitions of “iss” not just “s”
/(\w+)\s(\w+)/	saves first word in $1 and second in $2
/(\w+)/g	global match, matches all non-overlapping instances

Populate an array or hash with the results of global match

```
@array = ($string =~ /(\w+)/g);
%hash = ($string =~ /(\w+)/g);
```

Review of Session 7

Substitutions

```
$string =~ s/Hello/subst/;
```

Global, case-insensitive substitution:

```
$string =~ s/Hello/subst/gi;
```

Using capture groups:

```
$string =~ s/(\w+)/$1$1/g;
```

Non-destructive substitution:

```
$copy = ($original =~ s/world/Ithaca/ir);
```

Review of Exercise

1. Using our trusty random sequence generator, create a 9000 base pair length of sequence.
2. Using regular expressions find every instance of the sequence “ATGCAT” and delete it from the sequence
3. At each deletion, save the three base pairs on each side of the “ATGCAT” creating 2 arrays, one storing preceeding and one storing the trailing 3 base pairs
4. Print out the two arrays

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers


# Modify the sequence



# Record the preceeding and trailing trimers in separate arrays





# Print out the arrays
```

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers
@matches =

# Modify the sequence



# Record the preceeding and trailing trimers in separate arrays






# Print out the arrays
```

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers
@matches = ($sequence =~ m/                                        /ig);

# Modify the sequence



# Record the preceeding and trailing trimers in separate arrays







# Print out the arrays
```

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers
@matches = ($sequence =~ m/([acgt]{3})ATGCAT([acgt]{3})/ig);

# Modify the sequence



# Record the preceeding and trailing trimers in separate arrays





# Print out the arrays
```

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers
@matches = ($sequence =~ m/([acgt]{3})ATGCAT([acgt]{3})/ig);

# Modify the sequence
$matchcount =


# Record the preceeding and trailing trimers in separate arrays




# Print out the arrays
```

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers
@matches = ($sequence =~ m/([acgt]{3})ATGCAT([acgt]{3})/ig);

# Modify the sequence
$matchcount = ($sequence =~ s/                                  /gi);


# Record the preceeding and trailing trimers in separate arrays


 -






# Print out the arrays













-
```

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers
@matches = ($sequence =~ m/([acgt]{3})ATGCAT([acgt]{3})/ig);

# Modify the sequence
$matchcount = ($sequence =~ s/([acgt]{3})ATGCAT([acgt]{3})/$1$2/gi);


# Record the preceeding and trailing trimers in separate arrays









# Print out the arrays
```

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers
@matches = ($sequence =~ m/([acgt]{3})ATGCAT([acgt]{3})/ig);

# Modify the sequence
$matchcount = ($sequence =~ s/([acgt]{3})ATGCAT([acgt]{3})/$1$2/gi);
print "Found $matchcount matches\n";

# Record the preceeding and trailing trimers in separate arrays


# Print out the arrays
```

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers
@matches = ($sequence =~ m/([acgt]{3})ATGCAT([acgt]{3})/ig);

# Modify the sequence
$matchcount = ($sequence =~ s/([acgt]{3})ATGCAT([acgt]{3})/$1$2/gi);
print "Found $matchcount matches\n";

# Record the preceeding and trailing trimers in separate arrays
for($i=0;$i<=$#matches;$i+=2)
{


}

# Print out the arrays










-
```

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers
@matches = ($sequence =~ m/([acgt]{3})ATGCAT([acgt]{3})/ig);

# Modify the sequence
$matchcount = ($sequence =~ s/([acgt]{3})ATGCAT([acgt]{3})/$1$2/gi);
print "Found $matchcount matches\n";

# Record the preceeding and trailing trimers in separate arrays
for($i=0;$i<=$#matches;$i+=2)
{
        push @preceeding, $matches[$i];

}

# Print out the arrays
```

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers
@matches = ($sequence =~ m/([acgt]{3})ATGCAT([acgt]{3})/ig);

# Modify the sequence
$matchcount = ($sequence =~ s/([acgt]{3})ATGCAT([acgt]{3})/$1$2/gi);
print "Found $matchcount matches\n";

# Record the preceeding and trailing trimers in separate arrays
for($i=0;$i<=$#matches;$i+=2)
{
        push @preceeding, $matches[$i];
        push @trailing, $matches[$i+1];
}

# Print out the arrays
```

Review of exercise (/home/jarekp/perl_07/exercise7.pl)

```
# Find and save the trimers
@matches = ($sequence =~ m/([acgt]{3})ATGCAT([acgt]{3})/ig);

# Modify the sequence
$matchcount = ($sequence =~ s/([acgt]{3})ATGCAT([acgt]{3})/$1$2/gi);
print "Found $matchcount matches\n";

# Record the preceeding and trailing trimers in separate arrays
for($i=0;$i<=$#matches;$i+=2)
{
        push @preceeding, $matches[$i];
        push @trailing, $matches[$i+1];
}

# Print out the arrays
print "Preceding trimers:\n";
foreach $element (@preceeding)
{
    print "$element\n";
}
print "Trailing trimers\n";
foreach $element (@trailing)
{
    print "$element\n";
}
```

Example 1: FASTA file processing

Given:

- FASTA file
- List of requested sequence names (in a text file, one name per line)
- Short DNA motif (e.g., restriction site)

Objective:

Extract a subset of sequences which

- Are on the list of requested sequences **AND**
- Contain the requested DNA motif

Files in /home/jarekp/perl_08

- `extract_from_fasta.pl` (the script)
- `fasta_in.fa` (sample input file)
- `sequence_list.txt` (requested sequence names)

Fasta file

```
>jgi|Sorbi1|5257096|Sb01g000200
ATGCACCAAGCAGGGCAAGGCACTCCTGCTCCTACCGCCGCCGCGCCGAGCAGCCGCGCCCGCCGCATG
CATGCCGCACCCGGGACGGCTGCGCCGCAGCCCTCGCCAACACCCT
CTGCTCCCTCCTCCTGGGCCTCCT
CCTCATCGCCGCCGTCGTCGT
>jgi|Sorbi1|5257097|Sb01g000220
ATGGAGAAGCTGCCGACCTACGACCGCATGCGCCAGGGCATCCTCCGGCAGGCGCTCGCCGCCGGCGACC
AACAACAGAGCGGCGGCGTCGAGGTGGTGGACATGCATGAAGCTGGCCGGCGGCGACGGGGGCCGTGAACT
CTTGGAGCGCCTCTTCCAGGACGACAGCGAGCGATTCCTGCGCCGGCTCAGGGAC
>jgi|Sorbi1|5257098|Sb01g000245
atgctgtcactcacggtgtcgagctccctagcccagcccaacgtgtcgcctgcgtggctactgacatgtc
caaaaaatgttgacgccgtgatt
>jgi|Sorbi1|5257099|Sb01g000250
ATGGCGCACGTCCTGCACAGCTTCTGGCGACGCCACCGGTGGACGGCCCTGCGCTCCCTGCTCCTTGTCG
CGCTGCTCCACCGCCTCCATTTCTTGGCCTTCCTCGCCGCCTCCTCGCCCGTCTGGCTGCTCACCGCCTT
CCTTCTCGGTGCCGTCCTCGTCAACAGCGAACCCAACGTTCCCCTCGCAGCGGCATCAGAGGATGAGGAC
>jgi|Sorbi1|5257102|Sb01g000300
ATGGCGCGGACCAACTGGGAGGCCGATAAGATGCTCGACGTCTACATCTATGACTACCTGGTGAAGCGCA
ACCTGCAGGCCACCGCCAAGGCCTTCATCGCCGAGGGCAAGGTCGCCACCGACCCCGTCGCCATCGACGC
CCCCGGCGGCTTCCTCTTCGAGTGGTGGT
```

Requested sequences

```
jgi|Sorbi1|5257096|Sb01g000200
jgi|Sorbi1|5257098|Sb01g000245
jgi|Sorbi1|5257099|Sb01g000250
```

Requested motif (EcoT22I restriction site)

ATGCAT

Example 1: FASTA file processing

Outline of the algorithm

- Load names of requested sequences into memory (e.g., as **hash keys**)
- Read the input fasta file, line by line
  - A line starting with "**>**" signals new sequence and contains its name
    - analyze the **previous** sequence and decide whether to save it or not
      - Consult the hash of requested sequences, and look for the pattern
    - "remember" the new sequence name and set it as "current"
  - A line **NOT** starting with "**>**" is a part of the current sequence
    - collect the whole sequence line by line (to be analyzed after the next ">" found)
      - Do it only if the current sequence is among those requested (consult the hash)
- After the whole input file is scanned, the last sequence will still be in memory and needs to be analyzed (and saved, if needed) separately.

The command line (assuming all files are in the current directory):

```
./extract_from_fasta.pl fasta_in.fa  fasta_out.fa  sequence_list.txt ATGCAT
```

Example 1: FASTA file processing

Script overview

```
./extract_from_fasta.pl fasta_in.fa  fasta_out.fa  sequence_list.txt ATGCAT
```

Save the command line arguments:

```
$infile = $ARGV[0];   # path to the input fasta file
$outfile = $ARGV[1];  # path to the output fasta file
$selfile = $ARGV[2];  # path to the "\n"-delimited file
                      # with names of sequences to extract
$pattern = $ARGV[3];  # pattern to look for, e.g., ATTGCC
```

We will save the requested sequence names as keys of a hash **`%selseq`**

```
# the sequence names will be keys of the hash %selseq;







close IN;

# Report the number of requested sequences
```

We will save the requested sequence names as keys of a hash **%selseq**

```
# the sequence names will be keys of the hash %selseq;
open(IN,$selfile);






close IN;

# Report the number of requested sequences


```

We will save the requested sequence names as keys of a hash **`%selseq`**

```
# the sequence names will be keys of the hash %selseq;
open(IN,$selfile);
while($seqname=<IN>)
{



}
close IN;

# Report the number of requested sequences


```

We will save the requested sequence names as keys of a hash **`%selseq`**

```
# the sequence names will be keys of the hash %selseq;
open(IN,$selfile);
while($seqname=<IN>)
{
        chomp $seqname;

}
close IN;

# Report the number of requested sequences
```

We will save the requested sequence names as keys of a hash **`%selseq`**

```
# the sequence names will be keys of the hash %selseq;
open(IN,$selfile);
while($seqname=<IN>)
{
        chomp $seqname;
        $selseq{$seqname} += 1;
}
close IN;

# Report the number of requested sequences


```

We will save the requested sequence names as keys of a hash **`%selseq`**

```
# the sequence names will be keys of the hash %selseq;
open(IN,$selfile);
while($seqname=<IN>)
{
        chomp $seqname;
        $selseq{$seqname} += 1;
}
close IN;

# Report the number of requested sequences
$numselseq = scalar (keys %selseq);
```

We will save the requested sequence names as keys of a hash **%selseq**

```
# the sequence names will be keys of the hash %selseq;
open(IN,$selfile);
while($seqname=<IN>)
{
        chomp $seqname;
        $selseq{$seqname} += 1;
}
close IN;

# Report the number of requested sequences
$numselseq = scalar (keys %selseq);
print "$numselseq sequences requested\n";
```

We will save the requested sequence names as keys of a hash **`%selseq`**

```
# the sequence names will be keys of the hash %selseq;
open(IN,$selfile);
while($seqname=<IN>)
{
        chomp $seqname;
        $selseq{$seqname} += 1;
}
close IN;

# Report the number of requested sequences
$numselseq = scalar (keys %selseq);
print "$numselseq sequences requested\n";
```

Note: if

```
defined($selseq{$seqname})
```

is true, then the sequence called `$seqname` is among those requested

Loop logic

```
                        # Open the output file for writing
                        # Open the input file
                        # Lines of each sequence will be stored in this table (as strings)
# Scan the input file
```

Loop logic

```
open(OUT,">$outfile");  # Open the output file for writing
                        # Open the input file
                        # Lines of each sequence will be stored in this table (as strings)
# Scan the input file
```

Loop logic

```
open(OUT,">$outfile");  # Open the output file for writing
open(IN,$infile);       # Open the input file
                        # Lines of each sequence will be stored in this table (as strings)
# Scan the input file
```

Loop logic

```
open(OUT,">$outfile");  # Open the output file for writing
open(IN,$infile);       # Open the input file
@sequence_lines = ();   # Lines of each sequence will be stored in this table (as strings)
# Scan the input file
```

Loop logic

```
open(OUT,">$outfile");  # Open the output file for writing
open(IN,$infile);       # Open the input file
@sequence_lines = ();   # Lines of each sequence will be stored in this table (as strings)
# Scan the input file
$seqname = "";
while($line = <IN>)
{




















}
```

Loop logic

```
open(OUT,">$outfile");  # Open the output file for writing
open(IN,$infile);       # Open the input file
@sequence_lines = ();   # Lines of each sequence will be stored in this table (as strings)
# Scan the input file
$seqname = "";
while($line = <IN>)
{
        chomp $line;
















}
```

Loop logic

```perl
open(OUT,">$outfile");  # Open the output file for writing
open(IN,$infile);       # Open the input file
@sequence_lines = ();   # Lines of each sequence will be stored in this table (as strings)
# Scan the input file
$seqname = "";
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started

        {









        }







}
```

Loop logic

```
open(OUT,">$outfile");  # Open the output file for writing
open(IN,$infile);       # Open the input file
@sequence_lines = ();   # Lines of each sequence will be stored in this table (as strings)
# Scan the input file
$seqname = "";
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started

        {





        }
        else
        {

           # line is not a header - collect the current sequence
        }
}
```

Loop logic

```
open(OUT,">$outfile");  # Open the output file for writing
open(IN,$infile);       # Open the input file
@sequence_lines = ();   # Lines of each sequence will be stored in this table (as strings)
# Scan the input file
$seqname = "";
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started

        {

              # check if *previous* sequence is worth saving
                if(defined($selseq{$seqname}))
                {

                     # check if pattern present; if yes, save the sequence
                }



        }
        else
        {

           # line is not a header - collect the current sequence
        }
}
```

Loop logic

```
open(OUT,">$outfile");  # Open the output file for writing
open(IN,$infile);       # Open the input file
@sequence_lines = ();   # Lines of each sequence will be stored in this table (as strings)
# Scan the input file
$seqname = "";
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started

        {

              # check if *previous* sequence is worth saving
                if(defined($selseq{$seqname}))
                {

                     # check if pattern present; if yes, save the sequence
                }

               $seqname = substr($line,1); # set the newly found sequence name as current

        }
        else
        {

           # line is not a header - collect the current sequence
        }
}
```

Loop logic

```perl
open(OUT,">$outfile");  # Open the output file for writing
open(IN,$infile);       # Open the input file
@sequence_lines = ();   # Lines of each sequence will be stored in this table (as strings)
# Scan the input file
$seqname = "";
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started

        {

              # check if *previous* sequence is worth saving
                if(defined($selseq{$seqname}))
                {

                     # check if pattern present; if yes, save the sequence
                }

               $seqname = substr($line,1); # set the newly found sequence name as current


        }
        else
        {

           # line is not a header - collect the current sequence
        }
}
close IN;           # Input files scanned - it can now be closed....
```

Saving the sequence for analysis

```
$seqname = "";
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started
        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {

                        # check if pattern present; if yes, save the sequence
                 }

               # set the newly found sequence name as current, re-set sequence memory
                $seqname = substr($line,1);

        }
        else
        {

               # line is not a header - collect the current sequence (each line as array element)




        }
}
# Input files scanned - it can now be closed....
close IN;
```

Saving the sequence for analysis

```
$seqname = "";  @sequence_lines = ();
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started
        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {

                        # check if pattern present; if yes, save the sequence
                 }

                # set the newly found sequence name as current, re-set sequence memory
                 $seqname = substr($line,1);

        }
        else
        {

                # line is not a header - collect the current sequence (each line as array element)




        }
}
# Input files scanned - it can now be closed....
close IN;
```

Saving the sequence for analysis

```perl
$seqname = "";  @sequence_lines = ();
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started
        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {

                        # check if pattern present; if yes, save the sequence
                 }

               # set the newly found sequence name as current, re-set sequence memory
                $seqname = substr($line,1);
                @sequence_lines = ();
        }
        else
        {

               # line is not a header - collect the current sequence (each line as array element)




        }
}
# Input files scanned - it can now be closed....
close IN;
```

Saving the sequence for analysis

```
$seqname = "";  @sequence_lines = ();
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started
        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {

                       # check if pattern present; if yes, save the sequence
                 }

               # set the newly found sequence name as current, re-set sequence memory
                $seqname = substr($line,1);
                @sequence_lines = ();
        }
        else
        {

               # line is not a header - collect the current sequence (each line as array element)




        }
}
# Input files scanned - it can now be closed....
close IN;
```

Saving the sequence for analysis

```
$seqname = "";  @sequence_lines = ();
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started
        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {

                        # check if pattern present; if yes, save the sequence
                 }

               # set the newly found sequence name as current, re-set sequence memory
                $seqname = substr($line,1);
                @sequence_lines = ();
        }
        else
        {

               # line is not a header - collect the current sequence (each line as array element)
                if(defined($selseq{$seqname}))
                {

                }
        }
}
# Input files scanned - it can now be closed....
close IN;
```

Saving the sequence for analysis

```
$seqname = "";  @sequence_lines = ();
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started
        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {

                       # check if pattern present; if yes, save the sequence
                 }

               # set the newly found sequence name as current, re-set sequence memory
                $seqname = substr($line,1);
                @sequence_lines = ();
        }
        else
        {

               # line is not a header - collect the current sequence (each line as array element)
                if(defined($selseq{$seqname}))
                {
                        push(@sequence_lines,$line);
                }
        }
}
# Input files scanned - it can now be closed....
close IN;
```

Analyzing the sequence

```
$seqname = ""; @sequence_lines = ();
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started

        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {

                       # check if pattern present; if yes, save the sequence










                 }

               # set the newly found sequence name as current, re-set sequence memory
                $seqname = substr($line,1);
                @sequence_lines = ();
        }
        else {# line is not a header - collect the current sequence }
}
# Input files scanned - it can now be closed....
close IN;
```

Analyzing the sequence

```
$seqname = ""; @sequence_lines = ();
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started

        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {
                     # check if pattern present; if yes, save the sequence













                 }
               # set the newly found sequence name as current, re-set sequence memory
                $seqname = substr($line,1);
                @sequence_lines = ();
        }
        else {# line is not a header - collect the current sequence }
}
# Input files scanned - it can now be closed....
close IN;
```

Analyzing the sequence

```perl
$seqname = ""; @sequence_lines = ();
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started

        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {
                      # check if pattern present; if yes, save the sequence

                        $sequence = join("",@sequence_lines);









                 }
               # set the newly found sequence name as current, re-set sequence memory
                $seqname = substr($line,1);
                @sequence_lines = ();
        }
        else {# line is not a header - collect the current sequence }
}
# Input files scanned - it can now be closed....
close IN;
```

Analyzing the sequence

```
$seqname = ""; @sequence_lines = ();
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started

        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {
                      # check if pattern present; if yes, save the sequence

                        $sequence = join("",@sequence_lines);
                        if($sequence =~ /$pattern/i)
                        {





                        }
                 }

                # set the newly found sequence name as current, re-set sequence memory
                 $seqname = substr($line,1);
                 @sequence_lines = ();
        }
        else {# line is not a header - collect the current sequence }
}
# Input files scanned - it can now be closed....
close IN;
```

Analyzing the sequence

```
$seqname = ""; @sequence_lines = ();
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started

        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {

                      # check if pattern present; if yes, save the sequence

                        $sequence = join("",@sequence_lines);
                        if($sequence =~ /$pattern/i)
                        {
                             print OUT ">$seqname\n";




                        }
                 }
                # set the newly found sequence name as current, re-set sequence memory
                 $seqname = substr($line,1);
                 @sequence_lines = ();
        }
        else {# line is not a header - collect the current sequence }
}
# Input files scanned - it can now be closed....
close IN;
```

Analyzing the sequence

```
$seqname = ""; @sequence_lines = ();
while($line = <IN>)
{
        chomp $line;
        if($line =~ m/^>/)   # line is a header - new sequence started

        {

                # check if *previous* sequence is worth saving
                 if(defined($selseq{$seqname}))
                 {
                       # check if pattern present; if yes, save the sequence

                         $sequence = join("",@sequence_lines);
                         if($sequence =~ /$pattern/i)
                         {
                              print OUT ">$seqname\n";
                              for($i=0;$i<=$#sequence_lines;$i++)
                              {
                                   print OUT "$sequence_lines[$i]\n";
                              }
                         }
                 }
               # set the newly found sequence name as current, re-set sequence memory
                $seqname = substr($line,1);
                @sequence_lines = ();
        }
        else {# line is not a header - collect the current sequence }
}
# Input files scanned - it can now be closed....
close IN;
```

The last sequence

```
# ...but the last sequence is still memory;

# needs to be checked and printed out
if(defined($selseq{$seqname}))
{
        $sequence = join("",@sequence_lines);
        if($sequence =~ /$pattern/i)
        {
                print OUT ">$seqname\n";
                for($i=0;$i<=$#sequence_lines;$i++)
                {
                        print OUT "$sequence_lines[$i]\n";
                }
        }
}

# Now we can close the output file!
close OUT;
```

Example 2: SAM file processing

Complete SAM format specification: http://samtools.sourceforge.net/SAM1.pdf

Fragment of SAM file generated by Bowtie2

Header lines

```
@HD	VN:1.0	SO:unsorted
@SQ	SN:1	LN:301354135
@SQ	SN:2	LN:237068873
@SQ	SN:3	LN:232140174
@SQ	SN:4	LN:241473504
@SQ	SN:5	LN:217872852
@SQ	SN:6	LN:169174353
@SQ	SN:7	LN:176764762
@SQ	SN:8	LN:175793759
@SQ	SN:9	LN:156750706
@SQ	SN:10	LN:150189435
@SQ	SN:UNKNOWN	LN:7140151
@SQ	SN:Pt	LN:140384
@SQ	SN:Mt	LN:569630
@RG	ID:CAUZHENG58_CORLJSD3AADIBAPE_6	SM:CAUZHENG58	LB:CAUZHENG58	PL:ILLUMINA
@PG	ID:bowtie2	PN:bowtie2	VN:2.0.2
3_tPcjnVHL221	163	1	2265	18	75M	=	2671	481
AGATATTAAAGGAGGACTGTCCATGATTGGTCTGTTCGAAATTTTCAGCTATGGGACACCACACATGGGGTTTCT	IIIIIIIIIIIIIIIIIIIIIIIII8IIIB=DIIIC7I-:+I@BI;*I-
G&>592(6+:*&('2('2)#+/",&4	AS:i:146	XS:i:146	XN:i:0	XM:i:1	XO:i:0	XG:i:0	NM:i:1	MD:Z:68A6	YS:i:142	YT:Z:CP
RG:Z:CAUZHENG58_CORLJSD3AADIBAPE_6
3_tPcjnVHL221	83	1	2671	18	75M	=	2265	-481
ATGTGGGCCGTGTGCCGGGGTCCCAGGAGCAATCGCTTGCAGATTGGTGGTGAAAGGCTTGCAAACATCTCCCAA	-&1*%-
"%/5$@&D&85+4>(/I97'G88>:I(I@:19IIIII2AIIIIIDIIIIHIIIIIIIIIIIIIIIIII	AS:i:142	XS:i:106	XN:i:0	XM:i:2	XO:i:0	XG:i:0	NM:i:2	MD:Z:4T1A68	YS:i:146
YT:Z:CP	RG:Z:CAUZHENG58_CORLJSD3AADIBAPE_6
1_nIaNlVHL221	99	1	6675	44	75M	=	7074	474
TTCCACAAAAGTTATCTGACGGGCACAGTTGGTGAGCTTTTCATTCAGGCTGATCTCATCAACATTGTGTTTTTG	IIIIIIIIIIIIIIIIIIIIIIIIIII4IIIIIII<IIIIIIHIIII+ID4II+I3I-
+I3+%0%4I;HI4ICIIF	AS:i:150	XS:i:88	XN:i:0	XM:i:0	XO:i:0	XG:i:0	NM:i:0	MD:Z:75	YS:i:146	YT:Z:CP	RG:Z:CAUZHENG58_CORLJSD3AADIBAPE_6
7_AQydnVHL221	163	1	6715	44	75M	=	7124	484
TCATTCAGGCTGATCTCATCAACATTGTATTTTTTATCACGGCGCTTCAGCAGATTCAAAAGTTGAACCTCCCTA
IIIIIIIIIIIIIIIIIIIIIIHIIII6HIIIIIIIII?;9ICII,G=>;?:@9<.<7D%<)6*5-).*+5+&(3,	AS:i:135	XN:i:0	XM:i:2	XO:i:0	XG:i:0	NM:i:2	MD:Z:28G5G40	YS:i:150
YT:Z:CP	RG:Z:CAUZHENG58_CORLJSD3AADIBAPE_6
```

……………………. E t c ………………………………………….

Example 2: SAM file processing

A few words about BAM files (nothing to do with perl)

Converting from BAM (binary) to SAM (text) format

write SAM to STDOUT

```
samtools view –h maize_tst.bam
```

write SAM to file filename.sam

```
samtools view –h maize_tst.bam > maize_tst.sam
```

Converting from SAM (text) to BAM (binary) format

read SAM from file

```
samtools view –Sb maize_tst.sam > maize_tst.bam
```

read SAM from STDIN

```
cat maize_tst.sam | samtools view –Sb - > maize_tst.bam
```

Example 2: SAM file processing

Given:
- Illumina reads alignment in BAM format (BAM = binary version of SAM)

Objective:
- Generate a filtered BAM file containing only reads such that
    - Primary alignment is better than the secondary one
    - Number of mismatches (w.r.t. reference) in a read is at most 2

Files in /home/jarekp/perl_08:

`filter_bam.pl`	(script)
`maize_tst.bam`	(test BAM file to be filtered)

- Command:

```
./filter_bam.pl maize_tst.bam maize_filtered.bam >& log
```

Excerpt from Bowtie2 manual

12.Optional fields. Fields are tab-separated. bowtie2 outputs zero or more of these optional fields for each alignment, depending on the type of the alignment:

AS:i:<N> Alignment score. Can be negative. Can be greater than 0 in --local mode (but not in --end-to-end mode). Only present if SAM record is for an aligned read.

XS:i:<N> Alignment score for second-best alignment. Can be negative. Can be greater than 0 in --local mode (but not in --end-to-end mode). Only present if the SAM record is for an aligned read and more than one alignment was found for the read.

YS:i:<N> Alignment score for opposite mate in the paired-end alignment. Only present if the SAM record is for a read that aligned as part of a paired-end alignment.

XN:i:<N> The number of ambiguous bases in the reference covering this alignment. Only present if SAM record is for an aligned read.

XM:i:<N> The number of mismatches in the alignment. Only present if SAM record is for an aligned read.

XO:i:<N> The number of gap opens, for both read and reference gaps, in the alignment. Only present if SAM record is for an aligned read.

XG:i:<N> The number of gap extensions, for both read and reference gaps, in the alignment. Only present if SAM record is for an aligned read.

NM:i:<N> The edit distance; that is, the minimal number of one-nucleotide edits (substitutions, insertions and deletions) needed to transform the read string into the reference string. Only present if SAM record is for an aligned read.

YF:Z:<S> String indicating reason why the read was filtered out. See also: Filtering. Only appears for reads that were filtered out.

MD:Z:<S> A string representation of the mismatched reference bases in the alignment. See SAM format specification for details. Only present if SAM record is for an aligned read.

SAM file processing: algorithm highlights

- Open the input BAM file by calling `samtools` and piping its output (SAM format) into perl input stream

```
open(IN,"samtools view -h $inbamfile |");
```

- Pipe the output stream (in SAM format) into to produce the output BAM file

```
open(OUT,"| samtools view -Sb - > $outbamfile");
```

- For each record (alignment)
  - Echo (i.e., output without any processing) all header lines

```
if($line =~ m/^@/) { print OUT "$line\n"; next; }
```

  - Extract values of AS, XS, and NM tags

```
if( $tags =~ /AS:i:(\d+)/ ) { $AS = $1; }
```

  - Skip records satisfying one or more of the following conditions:
    - AS is undefined (read is unmapped)
    - AS <= XS
    - XM > 2
  - Print surviving records to output stream piped into `samtools view –Sb -` command
- Print filter statistics to standard output stream

Exercises

Modify the script `extract_from_fasta.pl` to select sequences which

- Are on the list of requested sequences **OR**
- Contain a given DNA motif

Modify the script `filter_bam.pl` to

- filter out all alignments with indels (use XO and XG tags)
- Accept SAM input from STDIN and write output to STDOUT, so that the filtering command would be

```
samtools view –h maize_tst.bam | ./filter_bam.pl | samtools
view –Sb - > maize_filtered.bam
```