Příloha 1

Skript pro filtrování a třídění sekvencí se shodným tagem

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

# v. 0.01 M. Hřebíček, B. Peková, L. Mrázová


$filename = $ARGV[0];
$filename =~ /([A-Za-z0-9_]+)\.\w/;

$basename = "$1";
$summaryheader = "basename";
$summary = "$basename";

$conminseq = $ARGV[1];                # minimal number of sequences needed for valid consensus
$mislimit = $ARGV[2];                 # max numberf of mismatches in R1 primer

# counters
$linecounter = 0;                   # counts lines in fastq reads
$readcounter1 = 0;                  # R1 counts reads in fastq file
$readcounter2 = 0;                  # R2 counts reads in fastq file
$taggedcounter1 = 0;                # number of correctly tagged reads
$taggedcounter2 = 0;                # number of correctly tagged reads
$untaggedcounter1 = 0;              # R1 number of garbage reads
$untaggedcounter2 = 0;              # R2 number of garbage reads
$shortreadcounter1 = 0;             # R1 reads shorter than $readlenlimit
$shortreadcounter2 = 0;             # R1 reads shorter than $readlenlimit
$headercounter = 0;
$r1emptygenocounter = 0;            # R1 reads, where there we no difference from the consensus
$r2emptygenocounter = 0;            # R2 reads, where there we no difference from the consensus
$r1validbincounter = 0;               # how many tag bins have higher number of reads than minimal -
conminseq
$r2validbincounter = 0;               # how many tag bins have higher number of reads than minimal -
conminseq

# read parameters
$read1ref = "GTAAAACGACGGCCAGTGNNNNNNNNNNTTGGGTGCGTAACTTTGTCG";
$read1from1 = 0;                  # first constant part of the primer limits; counts from 0, not 1
$read1to1 = 17;
$read1from2 = 37;                 # second constant part of the primer limits; counts from 0, not 1
$read1to2 = 47;
# $mislimit = 2;                    # tolerated number of mismatches in primer sequnces
$taglength = 10;
$readlenlimit = 100;              # minimal read length
$r1trim = 0;                      # trim primer sequences
$r1trimlimit = 47;                # trim sequence from 0 to the limit
$r2trimlimit = 47;
$seqcycles = 260;                 # run length set at the sequencer
$r1gbaseqshift = 6814 - 38;       # add this to read position to get the position in GBA sequence

# difference calling
$r1diffrom = 48;                  # call differences only from this position in the read
$r1diffto = 120;                  # call differences only to this position in the read
$r2diffrom = 20;                  # call differences only from this position in the read
$r2diffto = 120;                  # call differences only to this position in the read

$r1difflimit = 0;                 # report difference between the ref and consensus only if there
are more diffs than this
$r2difflimit = 0;



# control variables
#$conminseq = 1;                # minimal number of sequences needed for valid consensus
$conresult = "";               # string sums up properties of consensus sequence

#  quality filtering parameters
#  int codes for ascii error codes and corresponding probabilities for Illumina 1.18+ and Phred+33
%asciip = qw(33 1 34 0.79433 35 0.63096 36 0.50119 37 0.39811 38 0.31623 39 0.25119 40 0.19953 41
0.15849 42 0.12589 43 0.1 44 0.07943 45 0.0631 46 0.05012 47 0.03981 48 0.03162 49 0.02512 50
0.01995 51 0.01585 52 0.01259 53 0.01 54 0.00794 55 0.00631 56 0.00501 57 0.00398 58 0.00316 59
0.00251 60 0.002 61 0.00158 62 0.00126 63 0.001 64 0.00079 65 0.00063 66 0.0005 67 0.0004 68 0.00032
69 0.00025 70 0.0002 71 0.00016 72 0.00013 73 0.0001 74 0.00008);
```

```
$r1qlimit = 1;                  # number of expected errors in the
$r1qfrom = 47;                  # calculate expected error in the region from
$r1qto = 147;                   # calculate expected error in the region ending at

$r2qlimit = 1;                  # the same for read2
$r2qfrom = 20;
$r2qto = 120;

# length filtering parameters
$r1minlength = 249;
$r2minlength = 249;

# filtering by similarity to reference sequence parameters
$r1maxmismatch = 7;
$r1from = 47;
$r1to = 147;

$r2maxmismatch = 10;
$r2from = 20;
$r2to = 120;


# files
$fastq1 = "$basename.fastq";
$read1inhand = "READ1IN";
open ($read1inhand, $fastq1) or die "Could not open $fastq1:$!\n";

$readtagdist1 = $basename.".readtagdist1.dat";                                  #
distribution of reads per tag
open (RDIST1, ">$readtagdist1") or die "Could not open $readtagdist1:$!\n";

$readtagdist2 = $basename.".readtagdist2.dat";                                  #
distribution of reads per tag
open (RDIST2, ">$readtagdist2") or die "Could not open $readtagdist2:$!\n";

$read1other = $basename.".read1other.fq";
open (READ1OTHER, ">$read1other") or die "Could not open $read1other:$!\n";

$tagfile = "all_tags.txt";
open (TAGS, ">>$tagfile") or die "Could not open $tagfile:$!\n";

$gnuplotfile = "gnuplotcommands.txt";                                           # commands
for printing of charts by gnuplot
open (GPLOT, ">>$gnuplotfile") or die "Could not open $gnuplotfile:$!\n";

$tablefile = "table.tsv";                                                       # produces
table with run parameters
open (TABLE, ">>$tablefile") or die "Could not open $tablefile:$!\n";

$read2in = "$basename.fastq";
$read2in =~ s/_R1_/_R2_/;
$read2inhand = "READ2IN";
open ($read2inhand, "$read2in") or die "Could not open $read2in:$!\n";

$read2out = $basename."read2out.fq";
open (READ2OUT, ">$read2out") or die "Could not open $read2out:$!\n";

$read2other = $basename.".read2other.fq";
open (READ2OTHER, ">$read2other") or die "Could not open $read2other:$!\n";

$analysis = $basename.".analysis.txt";
open (ANAL, ">$analysis") or die "Could not open $analysis:$!\n";

$diffdotalign1 = $basename.".diffdotalign1.txt";                                 # reads
forming alignments with consensus bases differing from reference
open (DIFALIGN1, ">$diffdotalign1") or die "Could not open $diffdotalign:$!\n";

$diffdotalign2 = $basename.".diffdotalign2.txt";                                 # reads
forming alignments with consensus bases differing from reference
open (DIFALIGN2, ">$diffdotalign2") or die "Could not open $diffdotalign:$!\n";


# read reference sequence from plain text file
# NOTE tag bases have to to be written as Ns in the sequence for correct counting of differences
from the reference
$seqref = "delE10productB.txt";
open (SEQREF, "$seqref") or die "Could not open $seqref:$!\n";

# read reference sequence as plain text
```

```
$reference = "";
  while ($line = <SEQREF>) {
   chomp($line);
   $reference = $reference.$line;
  }
$reference = uc($reference);                              # convert to uppercase
$revreference = revcom($reference);                       # reverse and complement
$reference = substr($reference, 0, $seqcycles);           # cut reference to the length of the run
$revreference = substr($revreference, 0, $seqcycles);   # cut reference to the length of the run

# hashes
%r1tagged = ();                  # hash with sequences ; tag is the key, value is a reference to 2d
array with reads
%r1header = ();                  # hash: key is the shortened header, value is the tag
%r2tagged = ();                  # hash with sequences ; tag is the key, value is a reference to 2d
array with reads
%tag1genotype = ();              # hash holding tags as keys and string with R1 genotype summary as
values; used for matching genotypes with same tags
%r1variants = ();
%r2variants = ();
%tagnumreads = ();               # number of tags with the same number of reads; number of reads is
the key


# no changeable parameters below this line
*********************************************************************************

print ANAL "Run: $basename\n\nTolerated number of mismatches in primer sequences: $mislimit\nTrim :
$r1trim\nShort read length limit: $readlenlimit\nMinimal number of sequences needed for valid
consensus: $conminseq\n";
print ANAL "\nRead 1 file: $fastq1\n";

$summaryheader = $summaryheader."\tMax primer mismatch\tMin con seqs";
$summary = $summary."\t$mislimit\t$conminseq";


@read = ();

# read Read 1 fastq file into hashes
$endfile = 0;

do {{
   ($endfile, @read) = getfqread ($read1inhand);          # read one fastq read
    $readcounter1++;                                      # increase read counter

    if ( filter($read[1], $read[3], $reference, $r1minlength, $r1maxmismatch, $r1from, $r1to,
$r1qfrom, $r1qto, $r1qlimit) == 0) { next;}

    if (tagged($read[1]) == 1) {
    #process tagged reads
         $tag = substr($read[1],$read1to1 + 1, $taglength);
         print TAGS "$tag\n";
         $taggedcounter1++;
         if (length($read[1]) < $readlenlimit) {$shortreadcounter1++;}

         # add read to r1tagged hash; tag is the key, the value is array of arrays
         push @{$r1tagged{$tag}}, [ @read ];                 # pushing anonymous array reference to
the read into the array

         # add the part of the header (before the space) as a key and tag as value to %r1header hash
         if ($read[0] =~ /^(\S+)\s/) {                                    # collect non-white header
text before the space
         if (exists($r1header{$1})) {print ANAL "\nRead 1 Duplicate header $1\n";}
           else {$r1header{$1} = $tag;}
          }
      }
    else {
     # process untagged reads - print them to a fastq file
           print READ1OTHER "$read[0]\n$read[1]\n$read[2]\n$read[3]\n";
           $untaggedcounter1++;
         }

}} until ($endfile == 1);

print ANAL "Read 1 reads: $readcounter1\nRead 1 filtered tagged reads: $taggedcounter1\nRead 1 non-
tagged reads: $untaggedcounter1\nRead 1 reads shorter than $readlenlimit: $shortreadcounter1\nTag
number: ".scalar(keys(%r1tagged))."\n\n";

$summaryheader = $summaryheader."\tR1 reads\tR1 fitered tagged\tR1 nontagged reads\tTags";
```

```
$summary = $summary."\t$readcounter1\t$taggedcounter1\t$untaggedcounter1\t".scalar(keys(%r1tagged));

# call difference between the reference and consensus sequences
$taggedcounter1 = 0;
$diffcallcounter =0;

while(($tag, $reads) = each(%r1tagged)) {        # $reads hold a reference to 2D array with reads
(sharing the same tag)
   $taggedcounter1++;

  # Count how many tags  have 1, 2, 3 ,  etc. reads; Add numbers to a hash
  if (exists ($tagnumreads{@$reads})) {$tagnumreads{@$reads} = $tagnumreads{@$reads} + 1;}    # add
the number of reads in @reads to hash
   else {$tagnumreads{@$reads} =  1;}

   # Determine consensus
  ($cc,$conresult) = consensusmaker($reads, 250, 0, $conminseq);
  if ($conresult == 1) {
   $r1validbincounter++;
   ($diff, $diffc) = diffcaller($reference, $cc, $r1diffrom, $r1diffto);

   if ($diff =~ /\w+/) { $diff =~ s/\s+//;} else { $diff = " ";}

   $tag1genotype{$tag} = $diff.":";                    # add string with diffrences to the hash, tag
is the key.

   if ($diffc > $r1difflimit) {                    # if more than tolerated number of differences
     $diffcallcounter++;
     $qq = dottedconsensus($reads,$reference,$cc);
     print DIFALIGN1 "R1 $tag $diffc $diff\n$qq\n\n";
  }
 }
}
print ANAL "Called consensus for Read 1 reads ...\nRead 1 tagged reads: $taggedcounter1\nRead 1
consensus reads  with variant sequence: $diffcallcounter\n";

$summaryheader = $summaryheader."\tR1 con reads\tR1 con reads with variants\tR1 valid bins";
$summary = $summary."\t$taggedcounter1\t$diffcallcounter\t$r1validbincounter";

# Now print numbers of tags with different numbers of reads
print RDIST1 "##Reads_per_tag Tags Reads\n";
foreach $n (sort { $a <=> $b } keys (%tagnumreads)) { print RDIST1 "$n ".$tagnumreads{$n}."
".$n*$tagnumreads{$n}."\n";}

# Erase the $tagnumreads hash
%tagnumreads = ();

# process Read 2 reads
# read the fastq file into a hash with tags read from r1header hash as keys; same as Read1 only tags
come from the hash

$endfile = 0;
do {{
   ($endfile, @read) = getfqread ($read2inhand);
   # filter : seq .. sequence, qual .. quality line, ref .. reference seq, minlength .. minimal
length, maxmismatch .. max of tolerated mismatches,
   # from  .. mismatch check from base, to .. dtto, qfrom .. quality dtto, qto .. dtto, qlimit ..
limit on expected number of errors
   if ( filter( $read[1], $read[3], $revreference, $r2minlength, $r2maxmismatch, $r2from, $r2to,
$r2qfrom, $r2qto, $r2qlimit) == 0) {next;}
    $readcounter2++;                                          # increase read counter

   $read[0] =~ /^(\S+)\s/;                                   # collect part of header text before
the space
   if (exists($r1header{$1})) {                              # if the part of the header before the
space exists in header hash, get coresponding tag
    #process tagged reads
         $tag = $r1header{$1};
         #print "$tag\n";
         $taggedcounter2++;
         if (length($read[1]) < $readlenlimit) {$shortreadcounter2++;}
         # trim sequence and quality lines
         if ($r1trim == 1) {
           $read[1] = substr($read[1], $r2trimlimit + 1, length($read[1]) - $r2trimlimit-1);
# trim sequence
           $read[3] = substr($read[3], $r2trimlimit + 1, length($read[3]) - $r2trimlimit-1);
# trim quality
          }
```

```
        # add read to r2tagged hash; tag is the key, the value is array of arrays
        push @{$r2tagged{$tag}}, [ @read ];                # pushing anonymous array reference to
the read into the array
      }
    else {
     # process untagged reads - print them to a fastq file
           print READ2OTHER "$read[0]\n$read[1]\n$read[2]\n$read[3]\n";
           $untaggedcounter2++;
         }

}} until ($endfile == 1);

print ANAL "Read 2 reads: $readcounter2\nRead 2 tagged reads: $taggedcounter2\nRead 2 non-tagged
reads: $untaggedcounter2\nRead 2 reads shorter than $readlenlimit: $shortreadcounter2\nTag number:
".scalar(keys(%r2tagged))."\n\n";


$summaryheader = $summaryheader."\tR2 reads\tR2 fitered tagged\tR2 nontagged reads\tTags";
$summary = $summary."\t$readcounter2\t$taggedcounter2\t$untaggedcounter2\t".scalar(keys(%r2tagged));


# call differences between the reference and consensus sequences

$diffcallcounter =0;
while(($tag, $reads) = each(%r2tagged)) {        # reads hold a reference to 2D array with reads
(sharing the same tag)
  $taggedcounter2++;

  # Count how many tags  have 1, 2, 3 ,  etc. reads; Add numbers to a hash
  if (exists ($tagnumreads{@$reads})) {$tagnumreads{@$reads} = $tagnumreads{@$reads} + 1;}    # add
the number of reads in @reads to hash
   else {$tagnumreads{@$reads} =  1;}

  ($cc,$conresult) = consensusmaker($reads, 250, 0, $conminseq);
  if ($conresult == 1) {             # if the number of reads in consensus is lower than conminseq
   $r2validbincounter++;
   ($diff, $diffc) = diffcaller($revreference, $cc, $r2diffrom, $r2diffto);

   if ($diff =~ /\w+/) { $diff =~ s/\s+//;} else { $diff = " ";}

   if (exists($tag1genotype{$tag})) {$tag1genotype{$tag} = $tag1genotype{$tag}.$diff;}  # R2
diffcodes are separated by ":"
   else {print ANAL "Error: Tag $tag from r2tagged hash not found in tag1genotype\n";}

   if ($diffc > r2difflimit) {                                    # if more than tolerated number
of differences
     $diffcallcounter++;
     $qq = dottedconsensus($reads,$revreference,$cc);
     print DIFALIGN2 "R2 $tag $diffc $diff\n$qq\n\n";
   }
  }
}
print ANAL "Called consensus for Read 2 reads ...\nRead 2 tagged reads: $taggedcounter2\nRead 2
consensus reads with variant sequence: $diffcallcounter\n";
$summaryheader = $summaryheader."\tR2 con reads\tR2 con reads with variants\tR2 valid bins";
$summary = $summary."\t$taggedcounter2\t$diffcallcounter\t$r2validbincounter";

# Now print numbers of tags with different numbers of reads
print RDIST2 "##Reads_per_tag Tags Reads\n";
foreach $n (sort { $a <=> $b } keys (%tagnumreads)) { print RDIST2 "$n ".$tagnumreads{$n}."
".$n*$tagnumreads{$n}."\n";}


# sort and count all variants

while(($tag, $genotype) = each(%tag1genotype)) {
   ($r1,$r2) =  split(/:/,$genotype);
   #print "$genotype  $tag \n";


   if (($r1 eq "") || ($r1 eq " ") || ($r1 eq "  ")) { $r1emptygenocounter++;}
   else {
         @r1var = split(/-/,$r1);
         foreach my $v (@r1var) {
         if (exists($r1variants{$v})) { $r1variants{$v} = $r1variants{$v} + 1;}
         else {$r1variants{$v} = 1;}
        }
```

```
       }

   if (($r2 eq "") || ($r2 eq " ") || ($r2 eq "  ")) { $r2emptygenocounter++;}
   else {
         @r2var = split(/-/,$r2);
         foreach $v (@r2var) {
         if (exists($r2variants{$v})) { $r2variants{$v} = $r2variants{$v} + 1;}
            else {$r2variants{$v} = 1;}
         }
     }
 }

print ANAL "Reads without variants in genotype hash : R1 $r1emptygenocounter, R2
$r2emptygenocounter\n";

print ANAL "\n$summaryheader\n$summary\n";
print TABLE "$summary\n";

print ANAL "\nRead 1 variants:\n";
 foreach $q (sort {$r1variants{$a} <=> $r1variants{$b}} keys %r1variants) {
    print ANAL "$q\t".$r1variants{$q}."\t".r1genomic($q)."\n";
 }

print ANAL "\nRead 2 variants:\n";
foreach $q (sort {$r2variants{$a} <=> $r2variants{$b}} keys %r2variants) {
    print ANAL "$q\t".$r2variants{$q}."\t".r2genomic($q)."\n";
 }

 # print graphs using gnuplot
 print "gnuplot -c test.pg \"R1 - $basename\"  \"$readtagdist1\"  > R1$basename.png\n";
 system "gnuplot -c test.pg \"R1 - $basename\"  \"$readtagdist1\"  \> R1$basename.png";
 print "gnuplot -c test.pg \"R2 - $basename\"  \"$readtagdist2\"  > R2$basename.png\n";
 system "gnuplot -c test.pg \"R2 - $basename\"  \"$readtagdist2\"  \> R2$basename.png";

close $read1inhand;
close RDIST1;
close RDIST2;
close READ1OTHER;
close $read2inhand;
#close READ2OUT;
close READ2OTHER;
close SEQREF;
close ANAL;
close DIFALIGN1;
close DIFALIGN2;
close TAGS;
close GPLOT;
CLOSE TABLE;


# END ************************************************************************************************


sub  r1genomic {
  my ( $variant ) = @_ ;
  my $warning = "";
     $variant =~ /(\d+)([ACGTNacgtn])/;
  my $pos = $1;
  my $base = $2;
     if (($pos == 77) && ($base eq "C")) {$warning = "\tD409H";}
     $pos =  6767 + $pos;
     return "$pos$base$warning";
}

sub  r2genomic {
  my ( $variant ) = @_ ;
  my $warning = "";
     $variant =~ /(\d+)([ACGTNacgtn])/;
  my $pos = $1;
  my $base = $2;
     if (($pos == 34) && ($base eq "C")) {$warning = "\tV460V";}
     if (($pos == 48) && ($base eq "C")) {$warning = "\tA456P";}
     if (($pos == 83) && ($base eq "C")) {$warning = "\tL444P";}
     $pos =  6767 + $pos;
     $base =~  tr/ACGTNacgta/TGCANtgcan/;
     return "$pos$base$warning";
}

# filter ***********************************************************************************************
```

```
# filters sequence by various criteria, if sequencce passes the filter it returns 1, otherwise 0
# newly added filtering by quality
# seq .. sequence, qual .. quality, ref .. reference, minlength .. minimal length, maxmismatch ..
max of tolerated mismatches,
# from  .. mismatch check from base, to .. dtto, qfrom .. quality dtto, qto .. dtto
sub filter {
my ($seq, $qual, $ref, $minlength, $maxmismatch, $from, $to, $qfrom, $qto, $qlimit) = @_;

  $ result = 1;

  # filtering by length
  # if (length($seq) < $minlength) { $result = 0;}
  # filtering by similarity to reference sequence
  if (length($seq) < $to) { $to = length($seq);}                # if the read sequence is shorter
than the limit, cut the limit
  if (seqfit($ref,$seq,$from,$to,$maxmismatch) == 0) {$result = 0;}

  # filtering by quality
  # Expected number of errors is the sum of probabilities in the region

  if (length($qual) < $qto) { $qto = length($qual);}                # if the read sequence is
shorter than the limit, cut the limit
  my $psum = 0;
  # print "$qual\nqfrom $from qto $qto\n";
  for my $p (0 .. $qto - $qfrom) {
    my $ascode = ord(substr($qual, $p, 1));
    if (exists($asciip{$ascode})) { $q = $asciip{$ascode}; $psum = $psum + $q;}
# $z = substr($qual, $p, 1); print "$z \tcode $ascode \tint $q  pos $p \n"
    else {print ANAL "Error converting ascii quality code $ascode int $q  pos $p \n";}
   }
  if ($psum >= $qlimit) { $result = 0;}
  return $result;
}

# diffcaller ********************************************************************************
# compares reference and tested (i.e. consensus) sequences, returns a CIGAR-like summary (e.g.
145A170C200T).
# Calls only substitutions,not deletions and insertions.
# Added a range in which the differences are called
sub diffcaller {
   my ($ref, $tested, $from, $to) = @_;
   my $diffstring = "";
   my $diffcounter = 0;

   my $lastpos = 0;
   if (length($tested) <= length($ref)) {$lastpos = length($tested) - 1;} else { $lastpos =
length($ref) - 1;}  # strings start at 0
   if ($from > $lastpos) {$from = $lastpos;}
   if ($to > $lastpos) {$to = $lastpos;}

   for my $a ($from .. $to) {
      if ( substr($ref,$a,1) ne (substr($tested,$a,1))) {
       $diffstring = $diffstring.$a.substr($tested,$a,1)."-";
       $diffcounter++;
      }
    }
    $diffstring = substr($diffstring, 0, length($diffstring) - 1);     # remove the last extra '-'
    return ($diffstring." ", $diffcounter);
}

# consensusmaker ****************************************************************************
# iterates over a set of sequences with the same tag (received as a reference to the array of arrays
sharing the same tag)
# base that has most counts in the same position is added to the consensus sequence that is returned
# if there are less that $conminseq (usually 3 or more ) sequence reads in a given position, it
breaks and returns consensus
#  consensusmaker will return result code
sub consensusmaker {
  my ($readsarrayref, $len, $posindex, $conminseq) = @_;
    my $consensus = "";                                  # consensus sequence
    my $con = "";                                        # local storage for consensus
    my $result = "";
    my $percent = 0;
    my $pos = 0;
    my $lastpos = 0;
    my $sum = 0;
    my $valid = 1;
```

```
    # if low number of reads in array
    if (@$readsarrayref <= $conminseq)    {$result = $result."L".@$readsarrayref;}
# NOTE quick reporter that the number of reads is sufficient
    if (@$readsarrayref < $conminseq)    {$valid = 0;}
    # check length of sequences
    my $c = 0;
    foreach my $a (0 .. @$readsarrayref - 1) {

      if (length ($readsarrayref->[$a][1]) <= $readlenlimit) { $c++;}
    }
    $result = $result."S$c";

    # iterate over all positions in sequences
   POS : foreach $pos (0 .. $len) {                # for each position
      my $Acount = 0;
      my $Ccount = 0;
      my $Gcount = 0;
      my $Tcount = 0;
      my $othercount = 0;
      my $highest = 0;
      my $conreads = 0;
      $con = 'X';

      # iterate over sequences from reads and count bases
      RD : foreach my $a (0 .. @$readsarrayref - 1) {

        #print $readsarrayref->[$a][1]." ".length($readsarrayref->[$a][1])."\n";
        if ($pos > (length ($readsarrayref->[$a][1]) - 1)) { next RD;}
        if    (substr($readsarrayref->[$a][1],$pos,1) eq 'A') {$Acount++;}
        elsif(substr($readsarrayref->[$a][1],$pos,1) eq 'C') {$Ccount++;}
        elsif(substr($readsarrayref->[$a][1],$pos,1) eq 'G') {$Gcount++;}
        elsif(substr($readsarrayref->[$a][1],$pos,1) eq 'T') {$Tcount++;}
        elsif(substr($readsarrayref->[$a][1],$pos,1) =~ /\S/) {$othercount++;}  # if other non-white
character
        #print "$Acount + $Ccount + $Gcount + $Tcount + $othercount\n"
      }
     if    ($Acount > $highest) { $highest = $Acount; $con = 'A';}
     if($Ccount > $highest) { $highest = $Ccount; $con = 'C';}
     if($Gcount > $highest) { $highest = $Gcount; $con = 'G';}
     if($Tcount > $highest) { $highest = $Tcount; $con = 'T';}
     if($othercount > $highest) { $con = 'N';}

     #print "$con $highest ";print "$Acount + $Ccount + $Gcount + $Tcount + $othercount\n";
     $conreads = $Acount + $Ccount + $Gcount + $Tcount + $othercount;
     if ($conreads < $conminseq) { last POS;} #break if the bases are counted from less than
$conminseq reads
     #else {
        $consensus = $consensus.$con;                                    # append the base to
consensus string
        $sum = $sum + 100*$highest/$conreads;                           # sum percentage of
sequences that carry the consensus base
        $lastpos = $pos;
        $highest = 0;
     #}
   }
  #print "sum $sum pos $pos\n";
  my $perc = int($sum/($lastpos + 1));                                      # divide the sum of
percentages by the last position
  $result = $result."C$perc";                                             # append the percentage
to result
  #print "cons $consensus $result\n";
  return ($consensus, $valid);
}

# getfqread
*********************************************************************************************
# skips lines that do not start with @, then reads and chomps 4 lines into an array, returns 0 if
not eof and the array
# otherwise 1 and an empty array
sub getfqread {
 my ($fhandle) = @_;
 my $endfile = 0;
 my $linecounter = 0;
 my @read = ();
 my $line = "";

 while ($linecounter ne 4) {
   if (defined ($line = <$fhandle>)) {
```

```
     chomp ($line);
     if(($line =~ /^\@/) && ($linecounter == 0))  { $read[0]= $line; $linecounter++; next;}  #
header line
     elsif($linecounter==0)  {next;}
     elsif($linecounter == 1)  { $read[1] = $line; $linecounter++; next;}
# sequence
     elsif($linecounter == 2)  { $read[2] = $line; $linecounter++; next;}
# +
     elsif($linecounter == 3)  { $read[3] = $line; $linecounter = 4;}
# quality, last line in fastq record
   }
   else {$endfile = 1; last; }
 }
 return ($endfile, @read);
}


# tagged ***********************************************************************************************
# test if there there are correct primer sequences flanking the tag in read1
sub tagged{
   my ($test) = @_;
   if ((seqfit($read1ref, $test, $read1from1, $read1to1, $mislimit) == 1) &&
       (seqfit($read1ref, $test, $read1from2, $read1to2, $mislimit) == 1))
   {return 1;}  else {return 0;}
}

# seqfit ***********************************************************************************************
# Checks if the start of the sequence is (almost) identical with the primer,
# assumes two reference and tested strings are equal or almost equal and have the same starts
# Returns 1 (passed) or 0 (failed).
sub seqfit {
  my($reference,$tested,$from,$to,$mislimit) = @_;

  # sanity checks
  my $lntest = length($tested)-1;
  my $lnref = length($reference)-1;

  if (($from > $lntest) || ($to > $lntest))  { return 0;}  # sequence too short

  my $mismatches = 0;
  for my $a ($from..$to) {
    if (substr($reference,$a,1) ne substr($tested,$a,1)) {$mismatches++;}
   }
  if ($mismatches <= $mislimit) {return 1;}  else {return 0;}
}

sub dottedconsensus{
  my ($reftoarray, $reference,$consensus) = @_;
  my $output = "$reference\n";
  foreach my $a (0 .. @$reftoarray - 1) {

     foreach my $b (0 .. length($reftoarray->[$a][1]) - 1) {
       if (substr($reftoarray->[$a][1], $b, 1) eq (substr($reference, $b, 1))) { $output =
$output.".";}
         else { $output = $output.substr($reftoarray->[$a][1], $b, 1);}
       }
      $output = $output."\n";
    }
  for $a (0 .. length($consensus) - 1) {
     if (substr($consensus, $a, 1) eq (substr($reference, $a, 1))) { $output = $output."-";}
         else { $output = $output.substr($consensus, $a, 1);}
   }
  return $output;
}

sub revcom {
  my ($sequ) = @_;
  $sequ = reverse $sequ;
  $sequ =~ tr/ABCDGHMNRSTUVWXYabcdghmnrstuvwxy/TVGHCDKNYSAABWXRtvghcdknysaabwxr/;
  return $sequ;
}
```