```
#!/usr/bin/perl

##############################################################################
#                                                                            #
# File: primer_designer.pl                                                   #
# Author: Alex Poole                                                         #
# Email: alex.poole@ucdenver.edu                                             #
# Version: 0.01.02                                                           #
#                                                                            #
##############################################################################

# There will be future revisions to this script.  Contact Alex Poole at
# above email if you have problems.  Primer3 executable must already be
# installed on your machine and in your systems $PATH.

use strict;
use warnings;

# ____________________
#|                    |
#| Global Variables   |
#|____________________|
#

# path variables

my $PRIMER3EXECUTABLE = 'primer3_core';
my $BOULDER_IO_INPUT_FILE_PATH = 'primer3_input.txt';
my $PRIMER3OUTPUT_FILE = 'pr3_out.txt';
my $INPUT_FASTA_READS = 'Agk_reads.fna';
my $FINAL_OUTPUT = 'final_output.tsv';

# Primer3 variable hash which sets default parameter values:

my %primer3_default = (

    SEQUENCE_ID                         =>  "",                 # (string;
default empty)
    SEQUENCE_TEMPLATE                   =>  "",                 #
(nucleotide sequence; default empty)
    SEQUENCE_INTERNAL_EXCLUDED_REGION   =>  "",                 # (interval
list; default empty)
    SEQUENCE_TARGET                     =>  "",                 # (interval
list; default empty)
#   SEQUENCE_INCLUDED_REGION            =>  "",                 # (interval
list; default empty)
```

```
    PRIMER_TASK                         =>  "pick_pcr_primers",  # (string;
default pick_pcr_primers)
    PRIMER_OPT_SIZE                     =>  20,                  # (int;
default 20)
    PRIMER_MIN_SIZE                     =>  18,                  # (int;
default 18)
    PRIMER_MAX_SIZE                     =>  30,                  # (int;
default 27)
    PRIMER_MAX_NS_ACCEPTED              =>  0,                   # (int;
default 0) 0=don't allow ambiguous bases
    PRIMER_PRODUCT_SIZE_RANGE           =>  "60-800",            # (size
range list; default 100-300)
    PRIMER_PRODUCT_OPT_SIZE             =>  0,                   # (int;
default 0) no optimum size
    PRIMER_MIN_GC                       =>  30.0,                # (float,
default 20.0) percentage
    PRIMER_MAX_GC                       =>  80.0,                # (float;
default 80.0) percentage
    PRIMER_GC_CLAMP                     =>  2,                   # (int;
default 0)
    PRIMER_MAX_END_GC                   =>  5,                   # (int;
default 5)
    PRIMER_MIN_TM                       =>  58.0,                # (float;
default 57.0)
    PRIMER_OPT_TM                       =>  62.0,                # (float;
default 60.0)
    PRIMER_MAX_TM                       =>  65.0,                # (float;
default 63.0)
    PRIMER_PAIR_MAX_DIFF_TM             =>  2.0,                 # (float;
default 100.0)
    PRIMER_TM_FORMULA                   =>  0,                   # (int;
default 0) 0=Breslauer KJ,et al PNAS 83:4746-50 (1986) and Rychlik W, et
al. NAR 18:6409-12 (1990)
                                                                 #
1=SantaLucia JR. PNAS 95:1460-65 (1998) "recomended"
    PRIMER_MAX_SELF_ANY                 =>  8.00,                # (decimal,
9999.99; default 8.00)
    PRIMER_PAIR_MAX_COMPL_ANY           =>  8.00,                # (decimal,
9999.99; default 8.00)
    PRIMER_MAX_SELF_END                 =>  3.00,                # (decimal,
9999.99; default 3.00) 3'-end
    PRIMER_PAIR_MAX_COMPL_END           =>  3.00,                # (decimal,
9999.99; default 3.00) 3'-end
    PRIMER_MAX_POLY_X                   =>  4,                   # (int;
default 5)
```

```
    PRIMER_LOWERCASE_MASKING           =>  0,                  # (int;
default 0) 0=No, 1=Yes
    PRIMER_NUM_RETURN                  =>  1,                  # (int;
default 5)
    PRIMER_MISPRIMING_LIBRARY          =>  'simple.ref',       # (string;
default empty)
    PRIMER_MAX_LIBRARY_MISPRIMING      =>  10.00,              # (decimal,
9999.99; default 12.00)
    PRIMER_LIB_AMBIGUITY_CODES_CONSENSUS  => 0                 # (boolean;
default 1)
    );


=fasta2hashList takes a list of 454 read FASTA file names as arguments and
returns a list of
hash references corresponding to each FASTA file.  For each hash the name
of the
"uaccno" of sequences are the keys (without ">") and sequences are
corresponding values.  The
files must be in exact FASTA format--there is no file format checking.
=cut

sub fasta2hashList {
    my @fileList = @_;
    my @hashRefList = ();
    my $index = 0;
    for my $file (@fileList) {
        open(CURR_FILE, "<$file") or die "could not open file $file\n";
        my %curr_hash = ();
        my $curr_seq = '';
        my $curr_title = '';
        while (my $curr_line = <CURR_FILE>) {
            chomp($curr_line);
            if ((substr($curr_line, 0, 1)) eq '>') {
              if ($curr_line =~ m/uaccno=/) {
                $index = index($curr_line, 'uaccno=') + 7;
                $curr_title = substr($curr_line, $index);
                #print "$index $curr_title\n";
              }
              else {
                $curr_title = substr($curr_line, 1);
                $curr_title = (split(/ /, $curr_title))[0];
              }
              $curr_seq = '';
              $curr_hash{$curr_title} = '';
            }
```

```
            else {
                $curr_seq = $curr_seq . $curr_line;
                $curr_hash{$curr_title} = $curr_seq;
            }
        }
        push(@hashRefList, \%curr_hash);
        close(CURR_FILE);
    }
    return @hashRefList;
}

sub writeBoulderIOfile {
    my @fileList = @_;
    my $microsatHash = $fileList[0];
    my $fasta_seqs = $fileList[1];
    my $param_ref = $fileList[2];
    my $allRepeatTally = $fileList[3];
    my @primer3List = ();
    my @monoLengths = keys(%$microsatHash);
    my $textOut = "";

    my $seqID = "";
    my $maskedSeq = "";
    my $excludedRegions = "";
    #open(SAT_FASTA, ">allMicrosatIDs.fna");
    open(F_OUT, ">$BOULDER_IO_INPUT_FILE_PATH") or die "could not open file
$BOULDER_IO_INPUT_FILE_PATH\n";
    foreach my $seq_name (keys(%$fasta_seqs)) {
        my $seq = $fasta_seqs -> {$seq_name};
        my $seqInfo = (scanSeqAgainstHash($seq, $microsatHash,
@monoLengths))[0];
        updateTally($allRepeatTally, $seqInfo);
        #my $seqInfo = $seq_info_list[0];
        if ($seqInfo eq "") {
            $seqID = $seq_name;
        }
        else {
            $seqID = "$seq_name" . "_" . $seqInfo;
            @primer3List = getTargetRegionFromID($seqID);
            #$maskedSeq = (maskRegions($seq, $primer3List[1]))[0];
            $param_ref -> { 'SEQUENCE_ID' } = $seqID;
            #$param_ref -> { 'SEQUENCE_TEMPLATE' } = $maskedSeq;
            $param_ref -> { 'SEQUENCE_TEMPLATE' } = $seq;
            $param_ref -> { 'SEQUENCE_TARGET' } = $primer3List[0];
            $param_ref -> { 'SEQUENCE_INTERNAL_EXCLUDED_REGION' } =
$primer3List[1];
```

```
            foreach my $parameter (keys(%$param_ref)) {
                #print F_OUT "$parameter=$$param_ref{$parameter}\n";
                $textOut = $textOut . "$parameter=$$param_ref{$parameter}
\n";
            }
            #print F_OUT "=\n";
            $textOut = $textOut . "=\n";
            #print SAT_FASTA ">$seq_name\n$seq\n\n";
        }
    }
    print F_OUT "$textOut";
    #close(SAT_FASTA);
    close(F_OUT);
    return \$allRepeatTally;
}

sub getTargetRegionFromID {
    my @inList = @_;
    my $seqID = $inList[0];
    my @allValues = split(/_/, $seqID);
    shift(@allValues);
    my $start = $allValues[1];
    my $length = ($allValues[2] - $start);
    my $total_microsats = (scalar(@allValues) / 3);
    my $excluded_regions = "$start,$length";
    my $type = $allValues[0];
    my $target_region = "";
    my $tmpStart = 0;
    my $tmpLength = 0;
    my $index = 1;
    while ($index < $total_microsats) {
        $tmpStart = $allValues[$index * 3 + 1];
        $tmpLength = ($allValues[$index * 3 + 2] - $tmpStart);
        if($tmpLength > $length) {
            $type = $allValues[$index * 3];
            $start = $tmpStart;
            $length = $tmpLength;
        }
        $excluded_regions = $excluded_regions . " $tmpStart,$tmpLength";
        $index++;
    }
    $target_region = "$start,$length";
    return ($target_region, $excluded_regions, $type);
}

sub writeResults {
```

```
    my @inList =@_;
    my $pr3output = $inList[0];
    my $dataOutputFile = $inList[1];
    my $allMicrosatRef = $inList[2];
    my $totalRepeats = $inList[3];
    my %allMicrosats = %$allMicrosatRef;
    my $total_multiple = 0;
    my $total_seqs = 0;
    my $total_nt = 0;
    my $curr_seq = "";

    open(OUTPUT, ">$dataOutputFile");
    open(PR3IN, "<$pr3output");
    my $type = "";
    my $mono_size = 0;
    my $primersReturned = 0;
    my $lprimerSeq = "";
    my $rprimerSeq = "";
    my $numRepeats = 0;
    my $totalCompound = 0;
    my $totalBroken = 0;
    my $seqID = "";
    my $seqName = "";
    my $repsBetweenPrimers = "";
    while (my $curr_line = <PR3IN>){
        chomp($curr_line);
        if($curr_line eq '=') {
            $total_seqs++;
            if ($primersReturned > 0) {
                $repsBetweenPrimers =
getNumberRepeatsBetweenPrimers($curr_seq, $seqID, $lprimerSeq,
$rprimerSeq);
            }
            print OUTPUT "$seqName,$mono_size,$type,$numRepeats,
$primersReturned,$lprimerSeq,$rprimerSeq,$repsBetweenPrimers\n";
            #print  "$seqName\t$mono_size\t$type\t$numRepeats\t
$primersReturned\t$lprimerSeq\t$rprimerSeq\n";
            $allMicrosats{$type}[0]++;
            $allMicrosats{$type}[1]+= $primersReturned;
            $lprimerSeq = "";
            $rprimerSeq = "";
            $primersReturned = 0;
            $repsBetweenPrimers = "";
        }
        my @values = split(/=/, $curr_line);
        if((scalar(@values)) == 2) {
```

```
            if($values[0] eq "SEQUENCE_TEMPLATE") {
                $curr_seq = $values[1];
            }
            elsif ($values[0] eq "PRIMER_LEFT_0_SEQUENCE") {
                $lprimerSeq = $values[1];
            }
            elsif ($values[0] eq "PRIMER_RIGHT_0_SEQUENCE") {
                $rprimerSeq = $values[1];
            }
            elsif ($values[0] eq "SEQUENCE_ID") {
                $seqID = $values[1];
                $seqName = (split(/_/, $seqID))[0];
                my @seqTypeList = getLongestType($seqID);
                $type = $seqTypeList[0];
                $total_multiple += ($seqTypeList[1] - 1);
                $numRepeats = $seqTypeList[2];
                $mono_size = length($type);
                $totalCompound += $seqTypeList[3];
                $totalBroken += $seqTypeList[4];
            }
            elsif ($values[0] eq "PRIMER_PAIR_NUM_RETURNED") {
                $primersReturned = $values[1];
            }
        }
    }
    print OUTPUT "\n\n";
    print OUTPUT "Microsat,monomer length,total,longest,has primers\n";
    foreach my $CurrMicrosat (keys(%allMicrosats)) {
        my $eachSat = $allMicrosats{$CurrMicrosat}[0];
        my $eachprim = $allMicrosats{$CurrMicrosat}[1];
        my $all_repeats =$totalRepeats->{$CurrMicrosat}[0];
        $mono_size = length($CurrMicrosat);
        print OUTPUT "$CurrMicrosat,$mono_size,$all_repeats,$eachSat,
$eachprim\n";
    }

    print OUTPUT "\nCompound,Broken\n$totalCompound,$totalBroken\n";
    close(OUTPUT);
    close(PR3IN);
    return 0;
}

sub stripStr {
    my @input = @_;
    my $string = $input[0];
    $string =~ s/\A[\s\t\n\r\f]*//;
```

```
    $string =~ s/[\s\t\n\r\f]*$//;
    return $string;
}

sub getLongestType {
    my @inList = @_;
    my $seqID = $inList[0];
    my @allValues = split(/_/, $seqID);
    shift(@allValues);
    my $numValues = scalar(@allValues)/3;
    my $type = $allValues[0];
    my $firstType = $allValues[0];
    my $mono_size = length($type);
    my $repeats = 0;
    my $start = $allValues[1];
    my $stop = $allValues[2];
    my $length = ($stop - $start) + 1;
    my $tmpLength = 0;
    my $tmpStart = 0;
    my $tmpStop = 0;
    my $index = 1;
    my $compound = 0;
    my $broken = 0;
    my $curr_type = "";
    while ($index < $numValues) {
        $tmpStart = $allValues[($index * 3) + 1];
        $tmpStop = $allValues[($index * 3) + 2];
        $tmpLength = ($tmpStop - $tmpStart) + 1;
        $curr_type = $allValues[($index * 3)];
        if ($curr_type ne $firstType) {
            $compound = 1;
        }
        if ($length < $tmpLength) {
            $length = $tmpLength;
            $type = $allValues[$index * 3];
            $mono_size = length($type);
        }
        $index++;
    }
    if ( ($numValues > 1) && ($compound == 0)) {
        $broken = 1;
    }
    $repeats = int($length/$mono_size);
    return ($type, $numValues, $repeats, $compound, $broken);
}
```

```
sub getNumberRepeatsBetweenPrimers {
    my @inList = @_;
    my $seq = (@inList)[0];
    my $seqID = (@inList)[1];
    my $lPrimer = (@inList)[2];
    my $rPrimer = (@inList)[3];

    my $leftBookEnd = index($seq, $lPrimer);
    $rPrimer = (reverseComplement($rPrimer))[0];
    my $rightBookEnd = rindex($seq, $rPrimer);

    my @allValues = split(/_/, $seqID);
    shift(@allValues);
    my $totalMicrosats = scalar(@allValues)/3;

    my $currSat = $allValues[0];
    my $currStart = $allValues[1];
    my $currStop = $allValues[2];

    my $index = 0;
    my $totalRepeats = 0;

    while ($index < $totalMicrosats) {
        $currSat = $allValues[($index * 3)];
        $currStart = $allValues[($index * 3) + 1];
        $currStop = $allValues[($index * 3) + 2];
        if (($currStart >= ($leftBookEnd - 1)) && ($currStop <=
($rightBookEnd + 1))) {
            my $currLength = ($currStop - $currStart) + 1;
            my $currMonoSize = length($currSat);
            my $currRepeats = int($currLength/$currMonoSize);
            $totalRepeats += $currRepeats;
        }
        $index++;
    }
    return $totalRepeats;
}

sub maskRegions {
    my ($seq, $excluded) = ((@_)[0], (@_)[1]);
    my $maskedSeq = "";
    my @excludedList = split(/ /, $excluded);
    foreach my $tuple (@excludedList) {
        my $start = (split(/,/, $tuple))[0];
        my $length = (split(/,/, $tuple))[1];
        my $subst = "";
```

```
        for(1..$length) { $subst = $subst . "N"; }
        #my $first3rd = substr($seq, 0, $start - 1);
        #my $second3rd = substr($seq, $start - 1, $length, $subst);
        substr($seq, $start - 1, $length, $subst);
        #my $third3rd = substr($seq, $start + $length);
        #$seq = $first3rd . $second3rd . $third3rd;
    }
    return $seq;
}



sub createAllNmers {
    my $nmerLength = $_[0];
    my @bases = qw(A T C G);
    my @nmerList = ();
    my @posIndex = (0);
    for (my $i = 1; $i < $nmerLength; $i++) {
        push(@posIndex, 0);
    }
    while($posIndex[0] != -1 ) {
        my $newNmer = "";
        for (my $j = 0; $j < $nmerLength; $j++) {
            $newNmer = $newNmer . $bases[$posIndex[$j]];
        }
        push(@nmerList, $newNmer);
        @posIndex = updatePosIndex(@posIndex);
    }
    return @nmerList;
}

sub updatePosIndex {
    my @posIndex = @_;
    my $nmerLength = scalar(@posIndex);
    for (my $k = ($nmerLength - 1); $k >= 0; $k--) {
        if ($posIndex[$k] < 3) {
            $posIndex[$k]++;
            last;
        }
        elsif (($posIndex[$k] == 3) && ($k != 0)) {
            $posIndex[$k] = 0;
        }
        elsif (($posIndex[$k] == 3) && ($k == 0)) {
            return -1;
        }
    }
```

```
    return @posIndex;
}


sub makeGroup {
  my @inList = @_;
  my $curr_seq = $inList[0];
  my $seqSize = length($curr_seq);
  my $new_seq = "";
  my @group = ($curr_seq);
  for(my $i=0;$i < ($seqSize - 1); $i++){
    $new_seq = substr($curr_seq, ($seqSize - 1)) . substr($curr_seq, 0,
($seqSize - 1));
    push(@group, $new_seq);
    $curr_seq = $new_seq;
  }
  return @group;
}

sub removeFromArray {
  my @inList = @_;
  my $array_ref = $inList[0];
  my @array = @$array_ref;
  my $element = $inList[1];
  for (my $i=0;$i < scalar(@array); $i++) {
    if ($element eq $array[$i]) {
      splice(@array, $i, 1);
      $i--;
    }
  }
  return @array
}
sub groupAllNmers {
  my @list4mers = @_;
  my %groupHash = ();
  while (scalar(@list4mers) > 0) {
    my $leftInArray = scalar(@list4mers);
    my $curr4mer = shift(@list4mers);
    my @currGroup = makeGroup($curr4mer);
    $groupHash{$curr4mer} = \@currGroup;
    foreach my $item (@currGroup) {
      @list4mers = removeFromArray(\@list4mers, $item);
    }
  }
  return \%groupHash;
}
```

```
sub reverseComplement {
    my @seqList = @_;
    my $seqin = $seqList[0];
    my $rev_seq = reverse($seqin);
    $rev_seq =~ tr/ATCGatcg/TAGCtagc/;
    return $rev_seq;
}

sub removeRevCompGroups {
  my @inList = @_;
  my $hash_ref = $inList[0];
  my %big_hash = %$hash_ref;
  my $delete_self =0;

  my @allKeys = keys(%big_hash);
  my $index = 0;
  while(scalar(@allKeys) > 0) {
    $delete_self = 0;
    my $key = shift(@allKeys);
    my $seqLen = length($key);
    my $revKey = reverseComplement($key);
    foreach my $dictKey (keys(%big_hash)) {
      my $currListRef = $big_hash{$dictKey};
      my @currList = @$currListRef;
      for(my $i=0;$i<$seqLen;$i++) {
        if($revKey eq $currList[$i]) {
          $delete_self = 0;
          for(my $j=0;$j<$seqLen;$j++){
            if($key eq $currList[$j]){
              $delete_self = 1;
              last;
            }
          }
          if ($delete_self == 0) {
            $index++;
            delete $big_hash{$dictKey};
            @allKeys = removeFromArray(\@allKeys, $dictKey);
            last;
          }
        }
      }
    }
  }
  return \%big_hash;
}
```

```
sub createAllUniqueRepeats {
    my $nmerLength = $_[0];
    my @others2remove = ();
    foreach my $i (1..($nmerLength-1)) {
        if (($nmerLength % $i) == 0) {
            push(@others2remove, $i);
        }
    }
    my @nmers2remove = smallerGroups2remove(\@others2remove, $nmerLength);
    my @allNmers = createAllNmers($nmerLength);
    foreach my $currNmer (@nmers2remove) {
        @allNmers = removeFromArray(\@allNmers, $currNmer);
    }
    my $allGroups = (groupAllNmers(@allNmers))[0];
    my $finalGroupRef = (removeRevCompGroups($allGroups))[0];
    return $finalGroupRef;
}

sub smallerGroups2remove {
    my @inList = @_;
    my $toRemoveRef = $inList[0];
    my @nmers2remove = @$toRemoveRef;
    my $nmerLength = $inList[1];
    my $multiplier = 0;
    my @all2remove = ();
    foreach my $len (@nmers2remove) {
        $multiplier = $nmerLength / $len;
        my @allmers = createAllNmers($len);
        foreach my $monomer (@allmers) {
            my $currNmer = "";
            for (1..$multiplier) { $currNmer = $currNmer . $monomer; }
            push(@all2remove, $currNmer);
        }
    }
    return @all2remove;
}

sub createCountHash {
    my @microLengthList = @_;
    my %countHash = ();
    my $tmpHashRef;
    for my $microLength (@microLengthList) {
        $tmpHashRef = createAllUniqueRepeats($microLength);
        for my $satType (keys(%$tmpHashRef)){
```

```
            my $satGroupRef = $tmpHashRef -> {$satType};
            my @satGroup = @$satGroupRef;
            my $numSatsInGroup = scalar(@satGroup);
            my @valueList = (0, 0, $satType);
            foreach my $i (0 .. ($numSatsInGroup -1) ) {
                my $currSat = $satGroup[$i];
                my $revSat = (reverseComplement($currSat))[0];
                $countHash{$currSat} = [0, 0, $satType];
                $countHash{$revSat} = [0, 0, $satType];
            }
        }
    }
    return \%countHash;
}

sub scanSeqAgainstHash {
    my $seq = shift(@_);
    my $lookupHashRef = shift(@_);
    my @chunkSizes = @_;
    my $seqID = "";
    foreach my $chunk (@chunkSizes){
        my $partialID = (scanSeqWithWindow($seq, $chunk, $lookupHashRef))
[0];
        if ($partialID) {
            if ($seqID eq "") {
                $seqID = $partialID;
            }
            else {
                $seqID = $seqID . "_$partialID";
            }
        }
    }
    return $seqID;
}

sub createMicrosatHashCount {
    my @monoLengths = @_;
    my %hashOfMicrosatHashes = ();
    for my $mono (@monoLengths) {
        for my $offset (0 .. $mono - 1) {
            $hashOfMicrosatHashes{$mono}->{$offset} = ["", "",
(createCountHash($mono))[0]];
        }
    }
    return \%hashOfMicrosatHashes;
}
```

```
sub scanSeqWithWindow {
    my $seq = shift(@_);
    my $windowSize = shift(@_);
    my $masterHashRef = shift(@_);
    my $offsetHashRef = $masterHashRef->{$windowSize};
    my $seqID = "";
    my $seqLength = length($seq);
    my $thresholdLength = 12;
    my $minRepeats = $thresholdLength/$windowSize;

    if ($seqLength < ($windowSize * 2)) {
        return "";
    }
    else{
        my @offsetList = (0 .. ($windowSize - 1));
        my @runningRepeats = ();
        for (1 .. $windowSize ) { push(@runningRepeats, 0); }
        my %offsetHash = %$offsetHashRef;
        my $currSeq = "";
        my $prevSeq = "";
        my $except = "";
        my $cutOut = 0;
        my $offset = 0;
        my $pos = 0;
        while ($pos + $offset < $seqLength) {
            foreach $offset (@offsetList) {
                if ($pos + $offset < $seqLength) {
                    $currSeq = substr($seq, ($pos + $offset), $windowSize);
                    $prevSeq = $offsetHash{$offset}[1];
                    $offsetHash{$offset}[0] = $prevSeq;
                    $offsetHash{$offset}[1] = $currSeq;
                    eval {
                        $except = $offsetHash{$offset}[2]->{$currSeq}[2];
                    };
                    if ($except) {
                        if (($offsetHash{$offset}[2]->{$currSeq}[0]) > 0) {
                            $offsetHash{$offset}[2]->{$currSeq}[0]++;
                            $runningRepeats[$offset] = $offsetHash{$offset}
[2]->{$currSeq}[0];
                        }
                        elsif (($offsetHash{$offset}[2]->{$currSeq}[0]) ==
0) {
                            eval {
                                $except = $offsetHash{$offset}[2]-
```

```
>{$prevSeq}[2];
                            };
                            if ($except) {
                                if (($offsetHash{$offset}[2]->{$prevSeq}
[0]) >= $minRepeats) {
                                    $runningRepeats[$offset] =
$offsetHash{$offset}[2]->{$prevSeq}[0];
                                    $cutOut = 1;
                                }
                                $offsetHash{$offset}[2]->{$prevSeq}[0] = 0;
                            }
                            $offsetHash{$offset}[2]->{$currSeq}[1] = $pos +
$offset + 1;
                            $offsetHash{$offset}[2]->{$currSeq}[0] = 1;
                        }
                    }
                    else {
                        eval {
                            $except = $offsetHash{$offset}[2]->{$prevSeq}
[2];
                        };
                        if ($except) {
                            if (($offsetHash{$offset}[2]->{$prevSeq}[0]) >=
$minRepeats) {
                                $runningRepeats[$offset] =
$offsetHash{$offset}[2]->{$prevSeq}[0];
                                $cutOut = 1;
                            }
                            $offsetHash{$offset}[2]->{$prevSeq}[0] = 0;
                        }
                    }
                }
            }
            if ($cutOut == 1) {
                $seqID = (constructSeqID(\%offsetHash, $seqID, $windowSize,
$minRepeats, @runningRepeats))[0];
                $cutOut = 0;
                for my $q (0 .. $windowSize - 1) {
                    $runningRepeats[$q] = 0;
                    $offsetHash{$q}[2]->{$offsetHash{$q}[0]}[0] = 0;
                }
            }
            $offset = 0;
            $pos+=$windowSize;
        }
        $seqID = (constructSeqID(\%offsetHash, $seqID, $windowSize,
```

```
$minRepeats, @runningRepeats))[0];
    }
    return $seqID;
}

sub constructSeqID {
    my $offsetHashRef = shift(@_);
    my $seqID = shift(@_);
    my $windowSize = shift(@_);
    my $minRepeats = shift(@_);
    my @runningRepeats = @_;
    my $longestRepeat = 0;
    my $longestOffset = 0;
    my %offsetHash = %$offsetHashRef;
    my $ind = 0;
    for (0 .. ($windowSize - 1)) {
        if ($runningRepeats[$ind] > $longestRepeat) {
            $longestRepeat = $runningRepeats[$ind];
            $longestOffset = $ind;
        }
        $ind++;
    }
    if($longestRepeat >= $minRepeats) {
        my $currRepeat = $offsetHash{$longestOffset}[0];
        my $currStart = $offsetHash{$longestOffset}[2]->{ $currRepeat }[1];
        my $currType = $offsetHash{$longestOffset}[2]->{ $currRepeat }[2];
        my $repLength = $longestRepeat * $windowSize;
        my $stop = ($currStart + $repLength) - 1;
        if ($seqID eq ""){
            $seqID = "$currType\_$currStart\_$stop";
        }
        else {
            $seqID = $seqID . "_$currType\_$currStart\_$stop";
        }
    }
    return $seqID;
}

sub updateTally {
  my $allRepeats = shift(@_);
  my $seq_info = shift(@_);

  my $numRepeats = 0;

  if ($seq_info) {
    my @allInfo = split(/_/, $seq_info);
```

```
    $numRepeats = (scalar(@allInfo))/3;
    for ( my $i=0; $i < $numRepeats; $i++ ) {
      my $currRepeat = $allInfo[ ($i * 3) ];
      $allRepeats->{$currRepeat}[0]++;
    }
  }
  return \$allRepeats;
}



sub createMicrosatTallyHash {
    my @allMonos = @_;
    my @allTypes = ();
    my %tallyHash = ();
    foreach my $mono (@allMonos) {
        my $tmpRef = (createAllUniqueRepeats($mono))[0];
        my @tmpList = keys(%$tmpRef);
        push(@allTypes, @tmpList);
    }
    foreach my $type (@allTypes) {
        $tallyHash{$type} = [(0,0)];
    }
    return \%tallyHash;
}

$INPUT_FASTA_READS = $ARGV[0];
$FINAL_OUTPUT = $ARGV[1];


my $masterMicrosatHash = (createMicrosatHashCount(2,3,4))[0];
my $fasta_ref= (fasta2hashList($INPUT_FASTA_READS))[0];
my $microsatTally = (createMicrosatTallyHash(2,3,4))[0];
my $allmicrosatTally = (createMicrosatTallyHash(2,3,4))[0];
writeBoulderIOfile($masterMicrosatHash, $fasta_ref, \%primer3_default,
$allmicrosatTally);
my $commandLine = "primer3_core < $BOULDER_IO_INPUT_FILE_PATH >
$PRIMER3OUTPUT_FILE";
print `$commandLine`;
writeResults($PRIMER3OUTPUT_FILE, $FINAL_OUTPUT, $microsatTally,
$allmicrosatTally);
```