genefind.pl

```
#!/usr/bin/perl

print "\n\nEnter path for file to search: ";
$file = <>;
chop($file);

print "\n\nretrieving file $file...\n\n";

$/ = undef;
open(IN, $file) or die "\n\nError opening $file: $!\n";

$sequence = <IN>;
@gene = split(/CDS/, $sequence); #using CDS as splitter, cannot use
later!
foreach $gene(@gene) { #$gene[x] starts at>CDS
    next unless ($gene =~/gene/);
    $count++;

    if ($gene =~/gene="([A-Za-z0-9.0-9]+)"\s+\/note="(.+?)"/s){
        $gene_number = $1;
        $gene_name = $2;
    } #end if 1

    if ($gene =~/translation="(.+?)"/s){
      $translation = $1;
      $length = ($translation =~ tr/[A-Z]+//);
    } #end if 2

    if ($gene =~/gene\s+(\d+)..(\d+)/s) {
        $start = $1;
        $stop = $2;
        $count_forward++;
    } #end if 3

    if ($gene =~/gene\s+complement\((\d+)..(\d+)/s){
        $start = $1;
        $stop = $2;
        $count_reverse++;
    } #end if 4

    print "\n$count] gene name      :$gene_name\n";
    print "gene number          :$gene_number\n";
    print "start                :$start\n";
    print "stop                 :$stop\n";
    print "translation          :$translation\n";
    print "length               :$length\n";

} #end foreach

    if ($count == 0) {
         print "No genes found\n";
        }#end if
    else {
      print "\n$count gene(s) found.\n";
      print "\n$count_forward genes on sense strand found.\n";
        print "\n$count_reverse genes on complementary strand found.
\n\n\n";
    }#end else
```

cdsfind.pl

```
#!/usr/bin/perl
$/ = undef;

use Getopt::Long; #user input module
GetOptions("f|file=s" => \$file);
$value = $ARGV[0];
print "The value of argument 0 is $value\n\n";
print "Retrieving file $file...\n\n";
open(IN, $file) or die "\n\nError opening $file: $!\n";

$sequence = <IN>;
@exon = split(/gene\s+/, $sequence); #splitting arrays starting with
'gene'
foreach $exon(@exon){ #start 1
    next unless ($exon =~/CDS/);
    $count ++;
    if($exon =~/CDS\s+(complement)?\(join\((.+?)\)/s){ #start 2
      $sense = $1;
      $coordinates = $2; } #end 2
      if($sense =~/complement/){ #start 3
      $sense = "Reverse"; $reverse++; } #end 3
        else { #start 4
      $sense = "Forward"; $forward++; } #end 4

    print "\n\nGene $count on $sense strand:";
    $gene_count = 0; #resetting value for each exon

    @gene = split(/\,/, $coordinates); #splitting by comma separator
    foreach $gene(@gene){ #start 5
      $gene_count++;
      if($gene =~/(\d+..\d+)/s){ #start 6 #capturing (\d+..\d+)
format only and not simply printing out the array!
          $gene_coordinates = $1;
          print "\n$gene_count: $gene_coordinates";} #end 6
    } #end 5

    if ($exon =~/ORIGIN\s+(.+)/s) { #start 7
      $nucleotide = $1; } #end 7

    @basearray = split(/\s+\d/, $nucleotide); #splitting by spaces
between 10 consecutive nucleotide

    foreach $basearray(@basearray) { #start 8
      next unless ($basearray =~ /$value\s\d+/);
      $basecount++;
      print "$basearray[$basecount]\n";
    } #end 8

} #end 1
print "\n\n";
```

atgc6.pl

```
#!/usr/bin/perl
$/ = undef;

use Getopt::Long;
(GetOptions("f|filename=s" => \$file));

#$file = "/home/ubali/analysis/sequences/rice/94H10.seq";

open (IN, $file) or die "Cannot read $file: $!\n";

$line = <IN>;

$a = ($line =~tr/A//);
$t = ($line =~tr/T//);
$g = ($line =~tr/G//);
$c = ($line =~tr/C//);

$total = ($a + $t + $g + $c);
$gc = (($g + $c)/$total) * 100;

print "\nA         : $a\n";
print "T         : $t\n";
print "G         : $g\n";
print "C         : $c\n";

print "Total     : $total\n\n";
printf "GC content :%.1f%\n\n", $gc;
```

jobs.pl.

```
#!/user/bin/perl
$/ = undef;
$file = "/home/ubali/jobs/science_sept14.txt";
$search_term = "Bioinformatics";
open(IN, $file) or die "Error opening $file: $!\n";
$jobs_list = <IN>;
@jobs = split (/Title:/, $jobs_list);
print "Jobs matching keyword $search_term: \n\n";
foreach $job(@jobs){
    next unless $job =~ /$search_term/;
         if ($job =~ /$search_term/) {
           $count++;
        print "$count]          Title: $job\n";
       }
    }
```

jobs3.pl

```
#!/usr/bin/perl
$/ = undef;

use Getopt::Long;
use LWP::Simple;

(GetOptions("f|filename=s" => \$file, "s|search=s" =>
\$search_term));

open (IN, $file) or die "Error opening $file: $!\n";
$list = <IN>;

@jobs = split(/Title:/, $list);

foreach $job(@jobs) { #no semicolon for foreach statement

    next unless $job =~ /$search_term/i;
    if ($job =~ /$search_term.+URL:(.+)/s){ #no semicolon for if
statement
      $url = $1;
      $page = get($url);
      $count++;
      print "$count] Search criteria: $search_term<BR>";
      print "Job URL: $url\n";
      if ($page =~ /HW_JOB_AD_START>(.+)<HW_JOB_CONTROL/s){
          $ad = $1;
          print "<BR>$ad<BR>";
          print "=" x 70;
          print "<BR>";
          }
        print "$page\n";
    } #end if
} #end foreach

print "\n$count job(s) found\n";
```

jobs4.pl.

```
#!/usr/bin/perl
use LWP::Simple;

print "Enter path of file to parse: ";
$file = <>;
chop($file);

print "\n Enter search_term: ";
$search_term = <>;
chop($search_term);

print "Searching $file for $search_term...";

$/ = undef;
open(IN, $file) or die "Error opening $file: $!\n";
open(OUT, ">>$file.html") or die "Error opening $file.html: $!\n";

$list = <IN>;
@jobs = split (/Title:/, $list);
foreach $job(@jobs) { #no semicolon for foreach statement
    next unless $job =~ /$search_term/i;
    if ($job =~ /$search_term.+URL:(.+)/s){ #no semicolon for if
statement
      $url = $1;
      $page = get($url);
      $count++;
      print OUT "$count] Search criteria: $search_term<BR>";
      print OUT "Job URL: $url\n";
      if ($page =~ /HW_JOB_AD_START>(.+)<HW_JOB_CONTROL/s){
          $ad = $1;
          print OUT "<BR>$ad<BR>";
          print OUT "=" x 70;
          print OUT "<BR>";
      } #end if
    } #end if
} #end foreach

    if ($count == 0) {
      print "No jobs matching $search_term found\n";} #end if
    else {
      print "\n$count jobs(s) matching $search_term found.\nCheck
$file.html for the results.\n";}#end else
```

jobs.pl.

```
#!/usr/bin/perl
$/ = undef;
use LWP::Simple;

$file = $ARGV[0];
$search_term = $ARGV[1];

print "Searching $file for $search_term...";
open(IN, $file) or die "Error opening $file: $!\n";
open(OUT, ">>$file.html") or die "Error opening $file.html: $!\n";

$list = <IN>;
@jobs = split (/Title:/, $list);
foreach $job(@jobs) { #no semicolon for foreach statement
    next unless $job =~ /$search_term/i;
    if ($job =~ /$search_term.+URL:(.+)/s){ #no semicolon for if
statement
      $url = $1;
      $page = get($url);
      $count++;
      print OUT "$count] Search criteria: $search_term<BR>";
      print OUT "Job URL: $url\n";
      if ($page =~ /HW_JOB_AD_START>(.+)<HW_JOB_CONTROL/s){
          $ad = $1;
          print OUT "<BR>$ad<BR>";
          print OUT "=" x 70;
          print OUT "<BR>";
      } #end if
    } #end if
} #end foreach

    if ($count == 0) {
      print "No jobs matching $search_term found\n";} #end if
    else {
      print "\n$count jobs(s) matching $search_term found.\nCheck
$file.html for the results.\n";}#end else
```

asn51.pl

```
$/ = undef;

use Getopt::Long;
GetOptions("f|file=s" => \$file);

open(IN, $file) or die "Error opening $file: $!\n";

@array = split(/CDS\s+/, <IN>);

foreach $cds(@array) {
   next unless $cds =~ /join/;
   if ($cds =~ /(complement\()?join\((.+?)\)/s) {
      $sense = $1;
      $coord = $2;
      $gene++;
      if ($sense =~ /complement/) {$sense = "reverse";}
      else {$sense = "forward";}

   }

   @exons = split(/,/, $coord);
   print "\n\n";
   print "Gene $gene on $sense strand\n";
   foreach $exon(@exons) {
      $exon =~ s/^\s+//g;
      $count++;
      print "$count: $exon\n";
   }
   $count = 0;
}
```

assn52.pl

```
#!/usr/bin/perl
$/ = undef;

use Getopt::Long;
GetOptions("n|number=i" => \$resnum, "f|file=s" => \$file);
open(IN, $file) or die "Error opening $file: $! \n";

$record = <IN>;

if ($record =~ /bases 1 to (\d+)/) {$bacsize = $1;}

if ($resnum < 61  || $resnum == 61) {
   $limit = 61;
   if ($record =~ /ORIGIN (.+?) $limit/s) {$seq = $1; print "$seq\n";}
}

if ($resnum > 60) {
   $max = ($bacsize/60);
   if ($max =~ /(\d+)\./) {$integer = $1; $limitmax = (60*$integer);}
   for ($i = 0; $i < ($max+1); $i++) {
      $compare = (60*$i) + 1;
      if ($compare > $resnum) {
         $limit = (60*$i)+1;
       if ($limit > $limitmax) { getall(); } #sub-routine, see below
       if ($record =~ /ORIGIN (.+?) $limit/s) {$seq = $1; print "$seq\n";}
       last; # terminates loop
      }
      sub getall() {
         if ($record =~ /ORIGIN (.+)\/\//s) {$seq = $1; print "$seq\n";}
      }
   }
}
```

~~Gtest6~~ fast6_1.pl.

```
#!/usr/bin/perl
$/ = undef;

use Getopt::Long; #user input module
GetOptions("f|file=s" => \$file);

print "Retrieving file $file...\n\n";

open(IN, $file) or die "\n\nError opening $file: $!\n";

$exons = <IN>;
@input = split(/[a-z]+\=/, $exons);

print "\n\n";

foreach $input(@input) { # start 1
    @subarray = split(/\,/, $input);
    @revsort = reverse sort {$a <=> $b} @subarray;
    print "exons = @revsort\n";
} #end 1

print "\n\n";
```

Sort6-2.pl

```
#!/usr/bin/perl
$/ = undef;

@myarray = ("2:10","3:7","0:4","1:2");
print "\n\nUnsorted array: @myarray\n\n";

@myarray = sort (@myarray); #lexical sorting

print"Sorted by first field: @myarray\n";

foreach $myarray (@myarray) { #start 1
    ($x, $y) = split(/:/, $myarray);
    push (@secondnumber, "$y:$x"); #reversing coordinates
} #end 1

@secondnumber = sort {$a <=> $b} @secondnumber; #sorting reversed coordinates
numerically

foreach $secondnumber (@secondnumber) { #start 2
    ($x, $y) = split(/:/, $secondnumber);
    push (@sortedarray, "$y:$x"); #reassigning reversed coordinates to original
conformation
} #end 2

print"\nSorted by the second field: @sortedarray\n\n";
```

assn71.pl

```
#!/usr/bin/perl
$/=undef;

$file = "c:\\perl\\assn71input.txt";
open(IN,$file) or die "error opening $file :$!\n";
@gene=split(/\d+\]/,<IN>);

print "     Protein Name\t\t\t\t\t\tEC number\t\t\t\tAlternate names\n";
      print "-" x 120;


foreach $generec(@gene){
      next unless ($generec =~ /^.+\(EC/);
      if ($generec =~ /^(.+)\(EC\s+(\d+.+\d+)\)\s+\((.+)/)
      {
            $Pname =$1;
            $ECno =$2;
            $Aname1 =$3;
                  $count++;
            print " \n$count] $Pname\t\t\t\tEC $ECno\t\t\t\t";

      @record =split(/\s+\(/,$Aname1);
      foreach $record(@record){
            next unless ($record =~ /^.+\)/);
            if ($record =~ /(.+)\)/){
                  $word =$1;
                  print " $word,\n";
                  print " " x 100

            }
      }

      print"\n";
      }
}
```

ass72.pl

```
#!/usr/bin/perl

$/ = undef;

$file1 = "/home/ubali/Files/genes1.txt";
$file2 = "/home/ubali/Files/genes2.txt";

open(IN1, $file1) or die "Error opening $file1: $!\n";

@genes = split(/\n/, <IN1>);

foreach $gene(@genes) {
    next unless ($gene =~ /\d+/);
    if ($gene =~ /\d+\] (.+)/) {
      $name = $1;
      $count = 0;
        #$file2 = 'c:\genes2.txt';
        open(IN2, $file2) or die "Error opening $file2: $!\n";
        while($linex = <IN2>) {     #while loop runs till 2nd file is being read
         #regex for example: 0610005A21 riken_def FXYD domain-containing ion
transport regul..
         if ($linex =~ /\Q$name\E/) { print "Found: $name\n"; }
         else {print "Not found: $name\n";}
        }
      $number++;
    }
}
```

assn73.pl.

```
@array = qw(Ca++ K+ +++ ++++ ^++ #* *& Na+ OH- Cl- *.* ^ab* ^*ab*$%);
$string = join(" ",@array);
$string =~ tr/\s//d;
#print "$string";
if ($string =~ /([\w+^#*$%&-\.\s+]+)/)
{
      print "\n$1\n\n";
}
```

```
#!/usr/bin/perl
$/=undef;

$file = "/home/ubali/Files/genscan.txt";
die "Cannot Open File $file: $!\n" unless open (IN, $file); # Using
unless instead of else
$genscan = <IN>;

if ($genscan =~ /Tscr(.+)Suboptimal/s){ # Start if-1; Trimming input
file
    $input = $1; } # End if-1

print "\n\nGene\tStrand\tPromoter\tCoordinates\tGene size\tPolyA tail
\n\n";
print "-" x 75;

@genes = split(/.\d+\n\n\s\d+/,$input); # Splitting file into gene
record array

print "\n\nThe first element of this array is: $genes[0]\n\n";
#diagnostic print
print "The second element of this array is: $genes[1]\n\n";
#diagnostic print to see if array is created

foreach $genes(@genes) { # Start foreach-2

    next unless $genes =~ /Prom|Intr|PlyA|\s\+|\-\s+\d+\s+\d+\s+\d+/;

    if ($genes =~ /Prom\s(\+|\-)\s+(\d+)\s+(\d+)\s+\d+/) { # Start
if-3; Promoter coords
      $sense = $1;
      $begin = $2;
      $end = $3;
      $count++;
      print"\n\n$count\t$sense\t";

      if ($sense =~ /\+/) { # Start if-4
          print "$begin-$end\t$begin-";
      } # End if-4

      if ($genes =~ /\-/) { # Start if-5
          print "$end-$begin\t$begin-";
      } # End if-5

      else { # Start else-6
          print "----\t";
      } # End else-6
    } # End if-3

    if ($genes =~ /Term\s(\+|\-)\s+(\d+)\s+(\d+)\s+\d+/) { # Start
if-7; Gene coords
      $sense = $1;
      $begin_coordinate = $2;
      $end_coordinate = $3;

      if ($sense =~ /\+/) { # Start if-8
          print "$end_coordinate\t";
      } # End if-8

      if ($sense =~ /\-/) { # Start if-9
          print "\t$end_coordinate\t";
      } # End if-9
    } # End if-7


    @record = split(/\n/, $genes); # Splitting each record per line
```

```
    $gene_length = 0;
        foreach $record(@record) { # Start foreach-10
            next unless ($record =~ /\d+/);
            if ($record =~ /\d+.\d+\s+(Intr|Term|Init)\s+(\+|\-)\s+
\d+\s+\d+\s+(\d+).+/){ # Start if-11
            $size = $3;
            $gene_length += $size;
              } # End if-11
            } # End foreach-10

            print "$gene_length";

    if ($genes =~ /PlyA\s(\+|\-)\s+(\d+)\s+(\d+)\s+\d+/) { # Start
if-12; Poly A coords
      $plyA_sense = $1;
      $polyA_begin = $2;
      $polyA_end = $3;

      if ($plyA_sense =~ /\+/) { # Start if-13
          print "\t\t$polyA_begin - $polyA_end\n\n";
      } # End if-13

      else { # Start else-14
          print "\t\t$polyA_end - $polyA_begin\n\n";
      } # End else-14
    } # End if-12

} # End foreach-2
```

```
#!/usr/bin/perl
$/=undef;

use Getopt::Long;
GetOptions("f|file=s",\$file);
open(IN,$file) or die "Error opening $file: $!\n";

$file = <IN>;

@gene = split(/\n\n/,$file);

print "Gene\tStrand\tPromoter\t Coordinates\t\t Gene size\t polyA tail\n\n";

shift(@gene);
foreach $gene(@gene){
   next unless $gene =~ /^ \d+/;
   if ($gene =~ /(\+|\-)/){
      $sense=$1;
      $count++;
      print "$count\t$sense\t";
   }

   if($gene =~ /Prom\s+(\+|\-)\s+(\d+)\s+(\d+)/){
      $coor1 = $2;
      $coor2 = $3;

      if($sense =~ /\-/) { print"$coor2-$coor1\t"; }

      else { print"$coor1-$coor2\t";}
   }

   else { print"----\t\t"; }

   @line=split(/\n/,$gene);

   $Gsize=0;

   foreach $length(@line) {
      next unless ($length =~ /\d+/);
      if($length =~
/\d+.\d+\s+(Intr|Init|Term|Sngl)\s+(\+|\-)\s+\d+\s+\d+\s+(\d+).+/) {
         $len=$3;
         $Gsize+=$len;
      }
   }

   while(@line[0] !~
/\d+.\d+\s+((Intr|Init|Term|Sngl)\s+(\+|\-)\s+\d+\s+\d+.+)/) {
      shift(@line);
   }

   while(@line[-1]!~/\d+.\d+\s+((Intr|Init|Term|Sngl)\s+(\+|\-)\s+\d+\s+\d+.+)/)
{
      pop(@line);
   }

   $first=shift(@line);
   $last=pop(@line);

   if ($first=~/\d+.\d+\s+(Intr|Init|Term|)\s+(\+|\-)\s+(\d+)\s+(\d+).+/) {
      $co1=$3;
      $co2=$4;
   }

   if ($last=~/\d+.\d+\s+(Intr|Init|Term)\s+(\+|\-)\s+(\d+)\s+(\d+).+/) {
      $co3=$3;
      $co4=$4;
      print" $co1-$co4\t\t";
```

```
    }

    if ($gene=~/\d+.\d+\s+Sngl\s+(\+|\-)\s+(\d+)\s+(\d+).+/) {
       $co1=$2;
       $co2=$3;
       print " $co1-$co2\t\t";
    }

    print"$Gsize\t\t";

    if ($gene =~ /PlyA\s+(\+|\-)\s+(\d+)\s+(\d+)/){
       $copl1=$2;
       $copl2=$3;

       if($sense=~/\-/) {
          print"$copl2-$copl1\n";
       }

       else { print"$copl1-$copl2\n"; }

    }

    else { print "-----\n"; }
}
```

```
#!/usr/bin/perl
$/=undef;
use Getopt::Std;
use File::Basename;
use Cwd;

if ($#ARGV < 2) { #starts usage clause
        print << "USAGE";
                $0: requires the directory name & the file extension type\n
                Usage: $0 -d directory name -e extension_type (bak, pl
etc.)\n
USAGE
die "Please run program $0 again! $!\n\n";
} #ends usage clause

getopt("de");
$current_dir = cwd;

die "Error opening $opt_d: $!\n\n" unless opendir(DIR, $opt_d);

if ("$opt_d =~ $current_dir"){ #start if-1; for directory name = current
working directory
        print "\t\tThe current working directory is: $current_dir\n";
        print "\t\tPlease type another directory name:\n\n";
}#end if-1

print "The files in the directory $opt_d are:\n";

while(($filename = readdir(DIR))){ #start while-2
        ($filename, $dirname, $extn) = fileparse($filename, '\..*');
        if($extn =~ $opt_e){ #start if-3
                $count++;
        }# end if-3
        print"\t\t\t\t\t$filename\n";
} #end while-2
print " \nThere are $count total # of files in this directory with the
$opt_e extension\n\n";
```

```
#!/usr/bin/perl
$/=undef;
use Getopt::Std;
use File::Basename;
use Cwd;

if ($#ARGV < 1) { #starts usage clause
        print << "USAGE";
                \n$0: requires the directory name\n
Usage: $0 -d directory name\n
USAGE
die "Please run program $0 again! $!\n\n";
} #ends usage clause

getopt("d");
$current_dir = cwd;
$subdir_1 = "5902066";
$subdir_2 = "4506557";

if ("$opt_d = $current_dir"){ #start if-1; for directory name = current
working directory
        print "\nThe current working directory is: $current_dir\n";
        print "Please type another directory name:\n\n";
}#end if-1

die "Error opening $opt_d: $!\n\n" unless opendir(DIR, $opt_d);

print "The files in the directory $opt_d are:\n";

mkdir("$current_dir/$opt_d//$subdir_1");
mkdir("$current_dir/$opt_d//$subdir_2");

while(($filename = readdir(DIR))){ #start while-2
        ($filename, $dirname, $extn) = fileparse($filename, '\..*');
        print "The file being currently read is: $filename \n";
        if($filename =~ /5902066\_\d+/){ #start if-3
                system("mv /$opt_d/$filename$extn
/$opt_d/$subdir_1/$filename$extn");
                if($filename =~ /4506557\_\d+/){ #start if-4
                        system("mv /$opt_d/$filename$extn
/$opt_d/$subdir_2/$filename$extn");
                } #end if-4
        }# end if-3
} #end while-2
```

```
#!/usr/bin/perl
$/=undef;
use Getopt::Std;
use File::Basename;
use Cwd;

if ($#ARGV < 2) { #starts usage clause
        print << "USAGE";
                \n$0: requires the input directory path & the output
directory path\n
Usage: $0 -i input_dir -o output_dir\n
USAGE
die "Please run program $0 again! $!\n\n";
} #ends usage clause

getopt("io");
$current_dir = cwd;

if ("$current_dir =~ $opt_i"){ #start if-1; for directory name = current
working directory
        print "\nThe current working directory is: $current_dir\n";
        print "Please type another directory name:\n\n";
}#end if-1

die "Error opening $opt_i: $!\n\n" unless opendir(DIR, $opt_i);

if (!(-e "$current_dir/$opt_i/$opt_o")){ #start if-2
        mkdir("$current_dir/$opt_i/$opt_o");
        } #end if-2

while(($filename = readdir(DIR))){ #start while-3
        ($filename, $dirname, $extn) = fileparse($filename, '\..*');
        if($extn =~ ".out"){ #start if-4
                chdir "$opt_i";
                system(mv "$filename$extn
/home/Code/$opt_i/$opt_o/$filename$extn");
        }# end if-4
} #end while-3
```

```
$seq = "cccgggaaa";
print "Original sequence  = $seq\n";
$reverse = reverse $seq;
print "Reverse            = $reverse\n";
$seq =~ tr/[atgc]/[tacg]/;
print "Complement         = $seq\n";
$revcomp = reverse $seq;
print "Reverse complement = $revcomp";
```

```
$/ = undef;
use Getopt::Long;
GetOptions("file=s", \$file, "start=i", \$start,"end=i", \$end);

open(IN, $file) or die "Error opening $file: $!\n";
$cds = <IN>;
$cds =~ s/\d+//g;
$cds =~ s/\s+//g;

subseq($cds,$start,$end);

sub subseq($cds,$start,$end)
{
        $[=1;
        $seq = substr($cds,$start,(($end+1)-$start));
        print "$seq";
}

#use rnase.txt in C:\perl as input file
```

```
$/ = undef;

use Getopt::Long;
GetOptions("f|file=s" => \$file);

@enzymes = qw(ClaI EcoRI HindIII NdeI NheI SalI XhoI);
@sites = qw(atcgat gaattc aagctt catatg gctagc gtcgac ctcgag);

open(IN, $file) or die "Error opening $file: $!\n";
$cds = <IN>;
$cds =~ s/\d+//g;
$cds =~s/\s+//g;

for($i = 1; $i< $#enzymes; $i++) {
   search($sites[$i], $cds);
}

sub search($sites[$i],$cds)
{
        $count = 0;
        $ind = index($cds, $sites[$i]);
        if ( $ind != -1)
        {

                print "Cleavage sites for $enzymes[$i]: $ind";
                $count++;
        }

        $rind = rindex($cds, $sites[$i]);

        while($ind < $rind)
        {
                $seq = substr($cds,($ind+1),($rind - $ind));
                $index = index($seq, $sites[$i]);
                if ( $index != -1)
                {
                        print "$index\n";
                        $count++;
                }
                $ind++;
        }

        if ( $rind != -1)
        {
                print "$rind\n";
                $count++;
        }

        if(defined($count)) { print "The number of cleavage sites for
$enzymes[$i] is $count\n"; }

        else {  print "There are no cleavage sites in the sequence for
$enzymes[$i]\n"; }
}
```

```
undef $/;
$[ = 1;
use Getopt::Long;
GetOptions("f=s",\$file,"s=i",\$start,"e=i",\$end);
#end = length of sequence needed to be translated
open(IN,$file) or die "Error opening the file $file $!\n";
$cds = <IN>;
$cds =~ s/\d+//g;
$cds =~s/\s+//g;

$endpos = $start + 1;
$frame = 1;
$start--;

for($start;$start<=$endpos;$start++,$frame++)
{
        $gene = substr($cds, $start, $end);
        $length = length($gene);

        $gene =~ tr/[a-z]/[A-Z]/;
        print "\n\nFrame $frame:\n";
        print "$gene\n";
        $pos = 0;
        while ($pos <= $length)
        {
                $trip = substr($gene,$pos,3);

                if ($trip =~/^(A.+)/)
                {
                        $bega = $1;
                        beginA($bega);
                }
                if ($trip =~/^(T.+)/)
                {
                        $begt = $1;
                        beginT($begt);
                }
                if ($trip =~/^(G.+)/)
                {
                        $begg = $1;
                        beginG($begg);
                }
                if ($trip =~/^(C.+)/)
                {
                        $begc = $1;
                        beginC($begc);
                }
                $pos+=3;
        }
}


sub beginA($bega)
{
        if ($bega =~/(ATT|ATC|ATA)/){ print  "I  ";}
        if ($bega =~/(AAA|AAG)/){ print  "K  ";}
        if ($bega =~/(ATG)/){ print  "M  ";}
        if ($bega =~/(AAT|AAC)/){ print  "N  ";}
        if ($bega =~/(AGT|AGC)/){ print  "S  ";}
        if ($bega =~/(ACT|ACC|ACA|ACG)/){ print  "T  ";}
        if ($bega =~/(AGG|AGA)/){ print  "R  ";}
}
```

```
sub beginT($begt)
{
        if ($begt =~/(TGT|TGC)/){ print  "C  ";}
        if ($begt =~/(TTT|TTC)/){ print  "F  ";}
        if ($begt =~/(TGG)/){ print  "W  ";}
        if ($begt =~/(TAT|TAC)/){ print  "Y  ";}
        if ($begt =~/(TAA|TAG|TGA)/){ print  "---";}
        if ($begt =~/(TCT|TCC|TCA|TCG)/){ print  "S  ";}
        if ($begt =~/(TTA|TTG)/){ print "L  ";}
}

sub beginG($begg)
{

        if ($begg =~/(GAA|GAG)/){ print  "E  ";}
        if ($begg =~/(GTT|GTC|GTA|GTG)/){ print  "V  ";}
        if ($begg =~/(GAT|GAC)/){ print  "D  ";}
        if ($begg =~/(GCT|GCC|GCA|GCG)/){ print  "A  ";}
        if ($begg =~/(GGT|GGC|GGA|GGG)/){ print  "G  ";}

}

sub beginC($begc)
{
        if ($begc =~/(CCT|CCC|CCA|CCG)/){ print  "P  ";}
        if ($begc =~/(CAT|CAC)/){ print  "H  ";}
        if ($begc =~/(CGT|CGC|CGA|CGG)/){ print  "R  ";}
        if ($begc =~/(CTT|CTC|CTA|CTG)/){ print  "L  ";}
        if ($begc =~/(CAA|CAG)/){ print  "Q  ";}
}
```

```
#!/usr/bin/perl
$/=undef;

use Getopt::Std;

##############################################################################
############
##ARGUMENT FOR USER INPUT FORMAT##

if ($#ARGV < 3) { #starts usage clause
        print << "USAGE";
                \n$0: requires the input sequence along with the coding
coordinates\n
Usage: $0 -f input sequence file -s start coordinates -t termination
coordinate\n
USAGE
die "Please run program $0 again! $!\n\n";
} #ends usage clause
##############################################################################
############
##OPENING INPUT FILE & EXTRACTING NUCLEOTIDE SEQUENCE FROM GENEBANK RECORD##

getopt ("fst");
die "Error opening $opt_f: $!\n\n" unless open(IN, $opt_f);
$sequence = <IN>;
if ($sequence =~ /ORIGIN(.+)\/\//s){ # capture NCBI record start if-1
        $nucleotide = $1;
}#end if-1
print "\nSEQUENCE CAPTURED FROM GENBANK FORMAT IS: $nucleotide\n";

##############################################################################
############
##REMOVES WHITESPACE AND INTEGERS FROM GENBANK SEQUENCE RECORD##

$nucleotide =~ s/[0-9]+//g; #to remove all integers from the input sequence
$nucleotide =~ s/\s//g; #to remove all spaces
$end_cds = (($opt_t-$opt_s)+1); #to determine the length of sequence to be
extracted

##############################################################################
############
##EXTRACTS NUCLEOTIDE SEQUENCE FROM EXTRACTED SEQUENCE RECORD##

$[=1; #set query_sequence index at 1 for all subsequent operations
$query_sequence = substr($nucleotide, $opt_s, $end_cds); #function to
extract sequence using input coord
print "\nThe captured region is: \n$query_sequence\n";
$length = length($query_sequence); #to determine length of extracted
sequence
print "\nLength = $length\n";

##############################################################################
############
##CALCULATES %GC FOR EXTRACTED SEQUENCE##

$GC = ($query_sequence =~ tr/gc//); #to calculate %GC of input sequence
$percent_gc = ($GC/$length)*100;
print "\n%GC: $percent_gc\n\n";

##############################################################################
############
##CALCULATES FORWARD AND REVERSE PCR PRIMERS FOR EXTRACTED SEQUENCE##
```

```
$forward_primer = substr($query_sequence, 1, 20); #extracts 20 bases from
5'end
print "\PCR 20-mer 5'- 3' forward primer: $forward_primer\n";

$rev_primer_start = (($length - 20)+1);
$rp = substr($query_sequence, $rev_primer_start, 20); #extracts 20 bases
from 3' end
$rp =~ tr/ATGCatgc/TACGtacg/;
$reverse_primer = reverse $rp;
print "\nPCR 20-mer 5'-3' reverse primer: $reverse_primer\n";

##########################################################################
#############
##CALCULATES REVERSE AND REVERSE COMPLEMENT FOR EXTRACTED SEQUENCE##

$rev = reverse $query_sequence;
print "\nReverse:\n$rev\n\n"; #prints reverse sequence

$complement = $query_sequence;
$complement =~ tr/ATGCatgc/TACGtacg/;
print "Complement:\n$complement\n\n"; #prints Complement

$revcom = reverse $complement;
print "Reverse complement:\n$revcom\n\n"; #prints reverse complement

##########################################################################
##############
##CALCULATES RESTRICTION SITES FOR EXTRACTED SEQUENCE##

$AluI = "agct";
Restriction_Site_Find($AluI); #Invoking subrouting Restriction_Site_Find;
print "\n$site_count $_[0] AluI Site(s): @Enzyme\n";

$BpmI = "ctggag";
Restriction_Site_Find($BpmI); #Invoking subrouting Restriction_Site_Find;
print "\n$site_count $_[0] BpmI Site(s): @Enzyme\n";

$ClaI = "atcgat";
Restriction_Site_Find($ClaI); #Invoking subrouting Restriction_Site_Find;
print "\n$site_count $_[0] ClaI Site(s): @Enzyme\n";

$EcoRI = "gaattc";
Restriction_Site_Find($EcoRI); #Invoking subrouting Restriction_Site_Find;
print "\n$site_count $_[0] EcoRI Site(s): @Enzyme\n";

$EarI = "ctcttc";
Restriction_Site_Find($EarI); #Invoking subrouting Restriction_Site_Find;
print "\n$site_count $_[0] EarI Site(s): @Enzyme\n";

$GsuI = "ctggag"; #Isoschizomer for BpmI
Restriction_Site_Find($GsuI); #Invoking subrouting Restriction_Site_Find;
print "\n$site_count $_[0]GsuI Site(s): @Enzyme\n";

$HindIII = "aagctt";
Restriction_Site_Find($HindIII); #Invoking subrouting Restriction_Site_Find;
print "\n$site_count $_[0] HindIII Site(s): @Enzyme\n";

$NdeI = "catatg";
Restriction_Site_Find($NdeI); #Invoking subrouting Restriction_Site_Find;
print "\n$site_count $_[0] NdeI Site(s): @Enzyme\n";

$NheI = "gctagc";
Restriction_Site_Find($NheI); #Invoking subrouting Restriction_Site_Find;
```

```
print "\n$site_count $_[0] NheI Site(s): @Enzyme\n";

$SalI = "gtcgac";
Restriction_Site_Find($SalI); #Invoking subrouting Restriction_Site_Find;
print "\n$site_count $_[0] SalI Site(s): @Enzyme\n";

$XhoI = "ctcgag";
Restriction_Site_Find($XhoI); #Invoking subrouting Restriction_Site_Find;
print "\n$site_count $_[0] XhoI Site(s): @Enzyme\n";

#####-------------------------------------------------------------------------
---------####
sub Restriction_Site_Find(){ #start subroutine

        $[=1;
        $string_start_location = 1;
        $length = length($_[0]); #defining length of restriction site
        $site_count = 0; #setting RE site counter to 0
        @Enzyme = (); #ensuring array is empty at start of subroutine
        $Enzyme_last = rindex($query_sequence, $_[0]); #statement checked
with print value

        while ($string_start_location < $Enzyme_last){
                $Enzyme = index($query_sequence, $_[0],
$string_start_location);
                $string_start_location = ($Enzyme + $length);
                $site_count++;
                push (@Enzyme, $Enzyme);
        }#end while loop
        return (@Enzyme, $site_count);
}#end subroutine
####-------------------------------------------------------------------------
---------####
##############################################################################
##############
##TRANSLATES EXTRACTED SEQUENCE IN SIX READING FRAMES##

$[=1;
$Frame_1 = substr($query_sequence, 1);
$Frame_2 = substr($query_sequence, 2);
$Frame_3 = substr($query_sequence, 3);

print "\n\nFIRST FRAME TRANSLATION:\n";
print "-" x 24;
print "\n";
extract_codon($Frame_1);

print "\n\nSECOND FRAME TRANSLATION:\n";
print "-" x 25;
print "\n";
extract_codon($Frame_2);

print "\n\nTHIRD FRAME TRANSLATION:\n";
print "-" x 24;
print "\n";
extract_codon($Frame_3);
print "\n\n";

####-------------------------------------------------------------------------
--------####
sub extract_codon(){ #start subroutine

        $Frame_length = length($_[0]);
```

```
        $start_position = 1;

        while ($start_position < $Frame_length){ #extract codon triplets out
of each reading frame
                $codon = substr($_[0], $start_position, 3);
                $start_position += 3;
                translation_code($codon);
                print "$amino_acid";
        }# end while loop
} #end extract_codon subroutine
####-------------------------------------------------------------------------
-------####
sub translation_code(){ #start subroutine as listed in Tisdall, J.,
Beginning Perl for Bioinformatics

        if ($_[0] =~ /gc./i)            {$amino_acid = 'A'; return
$amino_acid}
        elsif ($_[0] =~ /tg[tc]/i)      {$amino_acid = 'C'; return
$amino_acid}
        elsif ($_[0] =~ /ga[tc]/i)      {$amino_acid = 'D'; return
$amino_acid}
        elsif ($_[0] =~ /ga[ag]/i)      {$amino_acid = 'E'; return
$amino_acid}
        elsif ($_[0] =~ /tt[tc]/i)      {$amino_acid = 'F'; return
$amino_acid}
        elsif ($_[0] =~ /gg./i)         {$amino_acid = 'G'; return
$amino_acid}
        elsif ($_[0] =~ /ca[tc]/i)      {$amino_acid = 'H'; return
$amino_acid}
        elsif ($_[0] =~ /at[tca]/i)     {$amino_acid = 'I'; return
$amino_acid}
        elsif ($_[0] =~ /aa[ag]/i)      {$amino_acid = 'K'; return
$amino_acid}
        elsif ($_[0] =~ /tt[ag]|ct./i) {$amino_acid = 'L'; return
$amino_acid}
        elsif ($_[0] =~ /atg/i)         {$amino_acid = 'M'; return
$amino_acid}
        elsif ($_[0] =~ /aa[tc]/i)      {$amino_acid = 'N'; return
$amino_acid}
        elsif ($_[0] =~ /cc./i)         {$amino_acid = 'P'; return
$amino_acid}
        elsif ($_[0] =~ /ca[ag]/i)      {$amino_acid = 'Q'; return
$amino_acid}
        elsif ($_[0] =~ /cg.|ag[ag]/i) {$amino_acid = 'R'; return
$amino_acid}
        elsif ($_[0] =~ /tc.|ag[tc]/i) {$amino_acid = 'S'; return
$amino_acid}
        elsif ($_[0] =~ /ac./i)         {$amino_acid = 'T'; return
$amino_acid}
        elsif ($_[0] =~ /gt./i)         {$amino_acid = 'V'; return
$amino_acid}
        elsif ($_[0] =~ /tgg/i)         {$amino_acid = 'W'; return
$amino_acid}
        elsif ($_[0] =~ /ta[tc]/i)      {$amino_acid = 'Y'; return
$amino_acid}
        elsif ($_[0] =~ /ta[ag]|tga/i) {$amino_acid = '_'; return
$amino_acid}
        else {print STDERR "Bad codon \"$codon\" !!\n";}
} #end subroutine
####-------------------------------------------------------------------------
------####
##############################################################################
############################
```