Chapter 5

Applying Perl

5.1 Introduction to chapter 5

In this chapter we will use the knowledge gathered during the first two weeks in order to solve some problems that may occur in the field of Bioinformatics. There will also be a lecture on how to use *references* in Perl. The following list is a summary of this chapter:

- Change of file formats
- References, objects and methods
- Searching in large text files
- Parsing Blast result files

During the lectures we will study programs (or part of programs) that solves the different tasks. We will look at both the structure of the program (i.e. how the author decided to logically solve the problem) and details in the Perl code.

5.2 Change of file formats

Pretend that you are performing a multiple sequence alignment. The result of such an alignment (using e.g. clustalw) can be a text file showing the alignment of the sequences. There are different formats for this text file. At a later step in your analysis of the alignments you may be faced with the problem of converting one text format to another. We will now look at the task of converting clustalw files (aln format) to phylip files (phylip format) and vise versa.

5.2.1 clustalw to phylip and phylip to clustalw

Here is an alignment of the 6 proteins `ACT1_FUGRU`, `ACT2_FUGRU`, `ACT3_FUGRU`, `5H1A_FUGRU`, `5H1B_FUGRU`, `5H1D_FUGRU`, shown in the "clustalw" format,

```
CLUSTAL W (1.82) multiple sequence alignment


ACT1_FUGRU      ----------------------MEDEIAALVVDNGSGMCKAGFAGDDAPRAVFPSIVGR 37
ACT2_FUGRU      ----------------------MDDEIAALVVDNGSGMCKAGFAGDDAPRAVFPSIVGR 37
ACT3_FUGRU      ----------------------MEDEVASLVVDNGSGMCKAGFAGDDAPRAVFPSIVGR 37
5H1A_FUGRU      MDLRATSSNDSNATSGYSDTAAVDWDEGENATGSGSLPDPELSYQIITSLFLGALILCSI 60
5H1B_FUGRU      -------MEGTNNTTGWT-----HFDSTSNRTSKSFDEEVKLSYQVVTSFLLGALILCSI 48
5H1D_FUGRU      -------MELDNNSLDYFSSN--FTDIPSNTTVAHWTEATLLGLQISVSVVLAIVTLATM 51
                                     *     .          .     :       :

ACT1_FUGRU      PRHQGVMVGMGQK-------DSYVGDEAQS--KRGILTLKYPIEHGIVTNWDDMEKIWHH 88
ACT2_FUGRU      PRHQGVMVGMGQK-------DSYVGDEAQS--KRGILTLKYPIEHGIVTNWDDMEKIWHH 88
ACT3_FUGRU      PRHQGVMVGMGQK-------DSYVGDEAQS--KRGILTLKYPIEHGIVTNWDDMEKIWHH 88
5H1A_FUGRU      FGNSCVVAAIALERSLQNVANYLIGSLAVTDLMVSVLVLPMAALYQVLNKWTLGQDICDL 120
5H1B_FUGRU      FGNACVVAAIALERSLQNVANYLIGSLAVTDLMVSVLVLPMAALYQVLNRWTLGQIPCDI 108
5H1D_FUGRU      LSNAFVIATIFLTRKLHTPANFLIGSLAVTDMLVSILVMPISIVYTVSKTWSLGQIVCDI 111
                  :  *:. :          :  :*. * :    .:*.:  .  : : . *   :   .

.
.
.

ACT1_FUGRU      PSIVHRKCF-- 375
ACT2_FUGRU      PSIVHRKCF-- 375
ACT3_FUGRU      PSIVHRKCF-- 375
5H1A_FUGRU      KKILRCKFHRH 423
5H1B_FUGRU      KKIIKCHFCRA 416
5H1D_FUGRU      QKLIK--FRR- 379
                 .:::
```

and here is the corresponding alignment in the phylip format,

```
     6    431
ACT1_FUGRU ---------- ---------- ---MEDEIAA LVVDNGSGMC KAGFAGDDAP
ACT2_FUGRU ---------- ---------- ---MDDEIAA LVVDNGSGMC KAGFAGDDAP
ACT3_FUGRU ---------- ---------- ---MEDEVAS LVVDNGSGMC KAGFAGDDAP
5H1A_FUGRU MDLRATSSND SNATSGYSDT AAVDWDEGEN ATGSGSLPDP ELSYQIITSL
5H1B_FUGRU -------MEG TNNTTGWT-- ---HFDSTSN RTSKSFDEEV KLSYQVVTSF
5H1D_FUGRU -------MEL DNNSLDYFSS N--FTDIPSN TTVAHWTEAT LLGLQISVSV

           RAVFPSIVGR PRHQGVMVGM GQK------- DSYVGDEAQS --KRGILTLK
           RAVFPSIVGR PRHQGVMVGM GQK------- DSYVGDEAQS --KRGILTLK
           RAVFPSIVGR PRHQGVMVGM GQK------- DSYVGDEAQS --KRGILTLK
           FLGALILCSI FGNSCVVAAI ALERSLQNVA NYLIGSLAVT DLMVSVLVLP
           LLGALILCSI FGNACVVAAI ALERSLQNVA NYLIGSLAVT DLMVSVLVLP
           VLAIVTLATM LSNAFVIATI FLTRKLHTPA NFLIGSLAVT DMLVSILVMP

.
.
.

           ASLSTFQQMW ISKQEYDESG PSIVHRKCF- -
           ASLSTFQQMW ISKQEYDESG PSIVHRKCF- -
           ASLSTFQQMW ISKQEYDESG PSIVHRKCF- -
           SLLNPIIYAY FN-KDFQSAF KKILRCKFHR H
           SLLNPIIYAY FN-KDFQSAF KKIIKCHFCR A
           SLINPVIYTV FN-DEFKQAF QKLIK--FRR -
```

The task is to write a program that can read a text file containing an alignment in one format and print (on the screen) the alignment using the other format. And vice versa. As usual when we are programming in Perl there many ways of solving this problem. I will present **one** of many possible solutions.

The program, called `convert-missing.pl` is divided into the following parts:

- Main program i.e. read command line arguments and load the alignment file to convert
- Subroutine parsing the Phylip format
- Subroutine parsing the ClustalW format
- Subroutine writing (to screen) a Phylip format
- Subroutine writing (to screen) a ClustalW format

Here is the main program.

convert-missing.pl - Main program

```
1
2  use strict;
3  our(%Seq, @Species);  # Global variables
4
5
6  #### The main program ####
7  open my $FH, '<', $ARGV[1] or die "Cannot open file $ARGV[1]: $!\n";
8  my @indata = <$FH>;
9  close $FH;
10
11 if( uc($ARGV[0]) eq 'P2C' ) {
12     ReadPhylip(@indata);
13     WriteClustalW();
14 }
15 elsif( uc($ARGV[0]) eq 'C2P' ) {
16     ReadClustalW(@indata);
17     WritePhylip();
18 }
19 else {
20     die "Error: Unknown conversion mode!\n";
21 }
22
```

convert-missing.pl - Main program

This program uses two global variables declared using the `our(%Seq, @Species);` statement. The file is read using the statement `my @indata = <$INFILE>`, which reads everything into the array `@indata`. The `uc()` function converts characters to upper case.

Here is the subroutine that reads a `phylip` file.

convert-missing.pl - sub ReadPhylip

```
1
2  sub ReadPhylip {
3
4      # Copy the supplied data to a local data array
5      my @data = @_;
6
7      # Use the first line to find the number of proteins
8      my ($nprot, $plen) = split ' ', shift @data;
```

```
9
10      # Read the first $nprot lines to get the species and the start of the seq
11      my @tmp;
12      for ( my $i = 0; $i < $nprot; $i++ ) {
13          ($Species[$i], @tmp) = split ' ', shift @data;
14          chomp @tmp;
15          $Seq{ $Species[$i] } = join '', @tmp;
16      }
17
18      # Continue to read lines of sequences
19      my $np = 0;
20      while ( my $line = shift @data ) {
21          next if ( $line =~ /^\s*$/ );
22
23          ### TO BE COMPLETED ###
24
25          $Seq{ $Species[$np] } .=  $line;
26
27          $np += 1;
28          if ( $np == $nprot ) {$np = 0;}
29      }
30  }
```

convert-missing.pl - sub ReadPhylip

To store the alignments a hash is used (i.e. the global variable `%Seq`), where a sequence is index by its name (i.e. the species). These names are also stored, in the correct order, in the array `@Species`. Some notations:

- The line `my @data = @_;` simply makes a copy of the supplied array and stores it in the local array `@data`.
- Throughout this routine the contents in `@data` is accessed using the `shift` function that chops off the first item of the array and returns it.
- The `split()` function is used here to split on whitespace.
- The "TO BE COMPLETED" will be left as an follow-up exercise.

Next, the corresponding routine that reads a `clustalw` file.

convert-missing.pl - sub ReadClustalW

```
1
2   sub ReadClustalW {
3
4       # Copy the supplied data to a local data array
5       my @data = @_;
6
7       # Skip starting blank lines
8       while ( $data[0] =~ /^\s*$/ ) {shift @data;}
9
10      # The first non-blank line should contain the word CLUSTAL
11      unless ( $data[0] =~ /CLUSTAL/ ) {
12          die "The input file does not contain the word CLUSTAL!\n";
```

```
    }
    shift @data;

    # Store the sequenses in a hash
    my %aux;
    while ( my $line = shift @data ) {
        next unless ( $line =~ /(-|\w)/ );

        my ($name, $seqtmp, $lentmp) = split ' ', $line;
        $Seq{$name} .= $seqtmp;

        # Find the name in correct order
        unless ( defined $aux{$name} ) {
            push @Species, $name;
            $aux{$name} = 1;
        }
    }
}
```

convert-missing.pl - sub ReadClustalW

Since the protein names are printed on every line of the alignment, it is easy to store the sequences in the `%Seq` hash. More specific comments:

- The line

  ```
  while( $data[0] =~ /^\s*$/ ) {shift @data;}
  ```

  is a short way of skipping blank lines in the beginning.

- There is a regexp for detecting lines that contains the alignments, namely `{/(-|\w)/}` which matches the character `'-'` or alphanumeric characters.

- The `%aux` hash is only used to get the correct order of the alignment when storing the names in the `@Species` array.

Next, the routine that writes a ClustalW format.

convert-missing.pl - sub WriteClustalW

```

sub WriteClustalW {

    # Write the ClustalW header
    print "CLUSTAL W (1.82) multiple sequence alignment\n\n\n";

    my $nprot = scalar @Species;
    my @aacount;
    while ( length($Seq{ $Species[0] }) > 0 ) {
        foreach my $i (0 .. $nprot-1) {
            printf "%10s\t", $Species[$i];
            $Seq{ $Species[$i] } =~ s/^(.{1,60})//;
            my $aas = $1;
            print "$aas";

```

```
16              # Count the number of non - in the aas
17              my $Naa = 0;
18              while ( $aas =~ /\w/g ) {
19                 $Naa += 1;
20              }
21              $aacount[$i] += $Naa;
22              if ( $aacount[$i] > 0 ) {
23                  print " $aacount[$i]\n";
24              }
25          }
26          print "\n\n";
27      }
28  }
```

convert-missing.pl - sub WriteClustalW

The task is here to write the names on each line followed by 60 amino acid symbols including gap characters. The line should end with a count of the number of amino acid letters written sofar.

- The line `$Seq{ $Species[$i] } =~ s/^(.{1,60})//;` may need some explanation. Here the task is "chop off", if possible, the first 60 characters of the sequence. This is accomplished with the above regexp. `s/^(.{q1,60})//` means up to 60 characters are replaced with nothing using the `s///` function. However the match is collected in `$1` using `()` in the regexp.

Finally, the routine that writes a Phylip format.

convert-missing.pl - sub WritePhylip

```
1
2   sub WritePhylip {
3
4       # Print the header line
5       printf "%d\t%d\n", scalar @Species, length($Seq{ $Species[0] });
6
7       # Now the rest of the sequences
8       my $PrintName = 1;
9       my $nprot = scalar @Species;
10      while ( length($Seq{ $Species[0] })  > 0 ) {
11
12          # The names
13          foreach my $i (0 .. $nprot-1) {
14              my $name = $Species[$i];
15
16              if( $PrintName == 1 ) {
17                  printf "%10s\t", $name;
18              } else {
19                  printf "%10s\t", ' ';
20              }
21
22              # The sequences
23              $Seq{$name} =~ s/^(.{1,50})//;
24              my $aas_tmp = $1;
25
```

```
26              ### TO BE COMPLETED ###
27
28              print "$aas\n";
29          }
30          print "\n";
31          $PrintName = 0;
32      }
33  }
```

convert-missing.pl - sub WritePhylip

5.2.2 Follow-up tasks 5-1

Download the files for this exercise:
`convert-missing.pl`, `clustalw1.aln`, `clustalw2.aln`, `phylip1.aln` and `phylip2.aln`.

and complete the following tasks:

1. Complete the first "TO BE COMPLETED". The `$line` scalar contains one line of the phylip file (without the names), e.g.

   ```
   K------LCY VALDFEQEMG TAASSSSLEK SYELPD---- --------GQ
   ```

   Use a regular expression to get rid of the spaces and newline characters in `$line`.

2. Complete the second "TO BE COMPLETED". This is the other way around. You have `$aas_tmp` which is a subsequence of length 50 and you should create `$aas` which is the same as `$aas_tmp` but space delimited after every 10 character. E.g.

   ```
   K------LCYVALDFEQEMGTAASSSSLEKSYELPD------------GQ
   ```

   should be

   ```
   K------LCY VALDFEQEMG TAASSSSLEK SYELPD---- --------GQ
   ```

3. Test the program using the files `clustalw1.align, phylip1.align` or `clustalw2.align, phylip2.align`. Note this program will not handle the conservation line appearing in the clustalw format.

4. Change the program so that one does not have to specify the conversion to make, i.e. the program should recognize that it reads a clustalw file and then convert it to a phylip file (and vice versa).

5. Add a subroutine to the program that finds the number of full matches in the multiple alignment. This number should be written at the end of the conversion. By a full match I mean a residue that is fully conserved among the proteins. Note: For the "clustal1.aln" alignment this number is 21.

Item 5 can be considered as "things to do if you have time".

5.3 References, objects and methods [1]

5.3.1 Creating references

What is a reference? This leads us to the concept of how Perl is storing values in variables. Each variable has a name and the address that corresponds to a piece of memory associated with it. Storing addresses is fundamental to references because a reference is a value that contains the location of another value. We call the scalar value that contains the memory address a *reference*. Lets look at the reference of a simple scalar variable,

```
$var = 'Hello world';
```

We can create a reference to this variable using the backslash operator.

```
$varref = \$var;
```

Both `$var` and `$varref` are scalars. Let's print them!

```
print "$var\n";
print "$varref\n";
```

results in the following output:

```
Hello world
SCALAR(0x815d7e4)
```

Here we can see that the reference is just a memory address that holds the value of `$var`. Why use references? In some circumstances it is very convenient and sometimes it is necessary. For example if you want to pass two arrays to a sub-routine, then we need references. We recall that everything passed to a sub-routine is stored in the `@_` array, which makes it difficult to pass two arrays. Another example is if we want to create hashes of arrays, then we also need references. Before we look at examples we need to know how to create references.

The backslash operator

You can create a reference to any named variable or subroutine with a backslash. Here are some examples:

```
$scalarref = \$foo;             # Reference to a scalar
$arrayref  = \@ARGV;            # Reference to an array
$hashref   = \%ENV;             # Reference to a hash
$constref  = \12341.42;         # Reference to a constant
$coderef   = \&myfunction;      # Reference to a sub-routine
($globref   = \*STDOUT;         # Reference to a glob)
```

[1] Inspiration of how to write this section is coming from "Programming Perl, third edition"

Anonymous data

In the examples just shown, the backslash operator makes a copy of a reference that is already in a variable name, with one exception; The 12341.42 is not referenced by a named variable, it is just a value. We can also create such anonymous arrays, hashes, and subroutines. For a reference to an anonymous array we use square brackets:

```
$anonarr1 = [1, 2, 4, 5, 6];
$anonarr2 = [1, 2, ['One', 'Two', 'Three', 'Four']];
```

The last one is a reference to an array, where the third element itself is a reference to another array. For hashes we use curly brackets to create the anonymous reference.

```
$anonhash1 = {                              # Reference to a hash
    'A'   => 'CGA',
    'C'   => 'TGC',
    'D'   => 'GAC'
    };
$anonhash2 = {                              # Reference to a hash of arrays
    'A'   => ['CGA', 'GCC', 'GCG', 'GCT'],
    'C'   => ['TGC', 'TGT'],
    'D'   => ['GAC', 'GAT']
    };
```

We can summarize the with the following table:

Reference to	Named	Anonymous
Scalar	`\$scalar`	
Array	`\@array`	`[ LIST ]`
Hash	`\%hash`	`{ LIST }`
Code	`\&function`	`{ CODE }`

5.3.2 Using references

There are many ways of creating references, as there are many way of using or *dereference* references. Before we look at small examples were we use references, we must learn how to dereference them.

Variable names

When you come across a scalar like `$amino`, you should be thinking "the scalar value of `amino`". This means that there is a `amino` entry in the "symbol table", and the $ character is a way of obtaining the scalar value behind the name. If what is inside is a reference, you can look inside that (dereferencing `$amino`) by prepending another $ character. Formulating

it in another way, you can replace the literal `amino` in `$amino` with a scalar variable that points to the actual referent. Here is an example,

```
$amino       = "PERLISFUN";
$scalarref   = \$amino;         # $scalarref is now a reference to $amino
$deref       = ${$scalarref};   # $deref is now "PERLISFUN"
```

We can of course use this way of dereferencing on both arrays and hashes. More examples,

```
$arrayref = \@aminos;                 # Make a reference to the array aminos
${$arrayref}[0] = "Gly";              # Set the first element of @$arrayref
push @{$arrayref}, "Ala";            # Add one element to  @$arrayref
@{$arrayref}[2..4] = qw/Val Leu Ile/; # Set several elements of @$arrayref

$hashref = \%ahash;                     # Make a reference to the hash ahash
%{$hashref} = ('GCA' => "A", 'GCC' => "A"); # Initialize whole hash
${$hashref}{'GCG'} = "A";                 # Set one key/value pair
```

Is is important to understand that dereferencing happens before any array or hash lookups. This is why it is important (at least in the beginning) to use curly braces. We can of course use shortcuts,

```
$deref = ${$scalarref};
is the same as
$deref = $$scalarref;

@{$arrayref}
is the same as
@$arrayref

%{$hashref}
is the same as
%$hashref
```

The use of braces is recommended. To help you understand this, note the important difference between `${$arrayref}[0]` and `${$arrayref[0]}` where the former means the first element of the array referred to by `$arrayref` and the latter, which is dereferencing the first element of the array named `@arrayref`. It is important to understand this!

The arrow operator

For references to arrays, hashes (or even subroutines), a third method of dereferencing involves the use of the `->` operator. Look at the following equivalent ways of dereferencing:

```
${$arrayref}[2] = "Val";         # The standard way
$$arrayref[2]   = "Val";         # The shortcut way
$arrayref->[2]  = "Val";         # This preferred "arrow way"
```

```
${$hashref}{'GCG'} = "A";          # The standard way
$$hashref{'GCG'}   = "A";          # The shortcut way
$hashref->{'GCG'}  = "A";          # This preferred "arrow way"
```

We can even use many of these arrow operators,

```
print $array[3]->{"Perl"}->[0];
```

You can see from this expression that the fourth element of `array` is a hash reference, and the value of the "Perl" entry in that hash is an array reference.

5.3.3 Examples of using references

Here we will show two examples of how to use references. The first one deals with subroutines and in the other one we create a hash of arrays.

Passing arrays to subroutines

ref2.pl

```
#! /usr/bin/perl -w
use strict;

#### Main part of the program
my @fasta = <>;

my $hashref = SearchFasta(\@fasta, ['TGG']);

my %res = %{$hashref};
foreach my $key ( sort keys %res ) {
    print "$key: $res{$key}\n";
}

sub SearchFasta {

    # Get the references from the argument array
    my ($f1ref, $subsref) = @_;

    # Dereference
    my @f1 = @{$f1ref}; # This makes a copy of the referenced array

    # Make a long string of the fasta sequence
    my $seq = join '', @f1;

    # Remove all comment lines
    $seq =~ s/>.*?\n//g;

    # Remove all new lines
    $seq =~ s/\n//g;

    # Loop over all substrings to search for
```

```
32      my %cnt;
33      foreach my $sub ( @{$subsref} ) {
34          $cnt{$sub} = ($seq =~ s/$sub/$sub/g);
35      }
36
37      # Return the hash of matches
38      return \%cnt;
39  }
```

ref2.pl

This small program contains one subroutine `SearchFasta` that takes two arguments, both references to arrays. The subroutine returns a reference to a hash. Can you figure out the purpose of the program? If we run the program on the first entry in the fasta file `ecoli.fasta` we get the following output:

```
1  TGG: 2044
```

Hash of arrays

In this little example we create a hash and where each value in the hash is an array. This can only be accomplished using references.

ref3.pl†

```
1   #! /usr/bin/perl -w
2   use strict;
3
4   # The translation table
5   my %codon;
6   $codon{'A'} = ['GCA', 'GCC', 'GCG', 'GCT'];
7   $codon{'C'} = ['TGC', 'TGT'];
8   $codon{'D'} = ['GAC', 'GAT'];
9   $codon{'E'} = ['GAA', 'GAG'];
10  $codon{'F'} = ['TTC', 'TTT'];
11  $codon{'G'} = ['GGA', 'GGC', 'GGG', 'GGT'];
12  $codon{'H'} = ['CAC', 'CAT'];
13  $codon{'I'} = ['ATA', 'ATC', 'ATT'];
14  $codon{'K'} = ['AAA', 'AAG'];
15  $codon{'L'} = ['CTA', 'CTC', 'CTG', 'CTT', 'TTA', 'TTG'];
16  $codon{'M'} = ['ATG'];
17  $codon{'N'} = ['AAC', 'AAT'];
18  $codon{'P'} = ['CCA', 'CCC', 'CCG', 'CCT'];
19  $codon{'Q'} = ['CAA', 'CAG'];
20  $codon{'R'} = ['AGA', 'AGG', 'CGA', 'CGC', 'CGG', 'CGT'];
21  $codon{'S'} = ['AGC', 'AGT', 'TCA', 'TCC', 'TCG', 'TCT'];
22  $codon{'T'} = ['ACA', 'ACC', 'ACG', 'ACT'];
23  $codon{'V'} = ['GTA', 'GTC', 'GTG', 'GTT'];
24  $codon{'W'} = ['TGG'];
25  $codon{'Y'} = ['TAC', 'TAT'];
26
27  # Print the translation table
28  foreach my $aa ( sort keys %codon ) {
```

```
29      print "$aa: ";
30      foreach my $nuc ( @{$codon{$aa}} ) {
31         print "$nuc ";
32      }
33      print "\n";
34  }
```

ref3.pl†

Note the square brackets when defining the hash `codon`. The expression `@{$codon{$aa}}` should be read as: The array referred to by the key `$aa` in the hash `%codon`.

5.3.4 Objects and methods

This very short section about objects and methods is meant as a very brief introduction to prepare you for the coming lectures about CPI.pm and the modules in the BioPerl project.

From "Programming Perl 3rd edition":

An *object* is a data structure with a collection of behaviors. Every object gets its behaviors by virtue of being an *instance* of a *class*. The class defines *methods*: behaviors that apply to the class and its instances. When the distinction matters, we refer to methods that apply only to a particular object as *instance methods* and those that apply to the entire class as *class methods*. But this is only a convention–to Perl, a method is just a method, distinguished only by the type of its first argument.

You can think of an instance method as some action performed by a particular object, such as printing itself out, copying itself, or altering one or more of its properties. Class methods might perform operations on many objects collectively or provide other operations that aren't dependent on any particular object.

Method invocation

This is how we invoke a method:

invocant->method(list)
invocant->method

E.g.

```
1  $seq1 = $gb->get_Seq_by_acc($ans);
```

5.3.5 Follow-up tasks 5-2

1. Make sure you understand the programs `ref2.pl` and `ref3.pl`.
2. Write a subroutine that takes two arrays as arguments and returns a hash. The hash is created from the two arrays such that the first element of the first array is used as a key and the first element of the second array is the corresponding value, and so on. Test your subroutine on some arrays.

3. Create an array and where each element of the array is a reference to a hash. Write a subroutine that takes such an array of hashes and display the values and keys of all hashes defined. Test your subroutine.

5.4 Searching in large text files

In this application, and the exercises that follows, we will use Perl to scan through large text files. The example below will utilize the Swiss-Prot database used in previous exercises and a cross-reference file, `pdb2sprot.txt`, linking PDB id's with Swiss-Prot ID/AC strings.

The task is to complete the missing information in the second column of the following table:

```
PDB        Organism
2YTE
1K46
1RJJ
113L
1S1J
2QKD
9LDB
3BW6
3MEJ
1S1S
7ZNF
1DU5
1C4Y
1FAD
1HY5
```

The "organism" information can be found in the Swiss-Prot database file and the link between an PDB ID and the Swiss-Prot ID can be found in the `pdb2sprot.txt`. Let's look at the two files!

5.4.1 The Swiss-Prot flat file

From the user manual of the Swiss-Prot file on can read[5]:

```
Swiss-Prot is an annotated protein sequence database. It was established in
1986 and maintained collaboratively, since 1987, by the group of Amos
Bairoch first at the Department of Medical Biochemistry of the University of
Geneva and now at the Swiss Institute of Bioinformatics (SIB) and the EMBL
Data Library (now the EMBL Outstation - The European Bioinformatics
Institute (EBI)). The Swiss-Prot Protein Knowledgebase consists of sequence
```

[5] http://www.expasy.ch/sprot/userman.html

entries. Sequence entries are composed of different line types, each with their own format. For standardization purposes the format of Swiss-Prot follows as closely as possible that of the EMBL Nucleotide Sequence Database.

Here is one entry from the Swiss-Prot file:

```
ID   Y491_PASMU              Reviewed;         127 AA.
AC   Q9CNE1;
DT   22-AUG-2003, integrated into UniProtKB/Swiss-Prot.
DT   01-JUN-2001, sequence version 1.
DT   10-AUG-2010, entry version 25.
DE   RecName: Full=Uncharacterized protein PM0491;
GN   OrderedLocusNames=PM0491;
OS   Pasteurella multocida.
OC   Bacteria; Proteobacteria; Gammaproteobacteria; Pasteurellales;
OC   Pasteurellaceae; Pasteurella.
OX   NCBI_TaxID=747;
RN   [1]
RP   NUCLEOTIDE SEQUENCE [LARGE SCALE GENOMIC DNA].
RC   STRAIN=Pm70;
RX   MEDLINE=21145866; PubMed=11248100; DOI=10.1073/pnas.051634598;
RA   May B.J., Zhang Q., Li L.L., Paustian M.L., Whittam T.S., Kapur V.;
RT   "Complete genomic sequence of Pasteurella multocida Pm70.";
RL   Proc. Natl. Acad. Sci. U.S.A. 98:3460-3465(2001).
CC   ---------------------------------------------------------------------
CC   Copyrighted by the UniProt Consortium, see http://www.uniprot.org/terms
CC   Distributed under the Creative Commons Attribution-NoDerivs License
CC   ---------------------------------------------------------------------
DR   EMBL; AE004439; AAK02575.1; -; Genomic_DNA.
DR   RefSeq; NP_245428.1; -.
DR   GeneID; 1243838; -.
DR   GenomeReviews; AE004439_GR; PM0491.
DR   KEGG; pmu:PM0491; -.
DR   NMPDR; fig|272843.1.peg.491; -.
DR   BioCyc; PMUL272843:PM0491-MONOMER; -.
PE   4: Predicted;
KW   Complete proteome.
FT   CHAIN         1    127       Uncharacterized protein PM0491.
FT                                /FTId=PRO_0000216293.
SQ   SEQUENCE   127 AA;  14589 MW;  A85EFFC5579E4184 CRC64;
     MQLVFSYIEH KSQVIPVCFW KENHQLHPLT GYLNDPMGGL NYFAFLDKVL SMLRDEDIQQ
     GDISSNSWGV EIHGDQVYFC FLFAQEDTSL HFALSRAVLI DILVLWLAFR SQKPVAGYQE
     VLSFAEA
//
```

In this application we will use the `ID`,`OS` and `SQ` lines.

5.4.2 The pdb2sprot.txt file

The `pdb2sprot.txt` file is a cross-reference between PDB ID's and (if possible) corresponding Swiss-Prot ID's. The first few lines of the file looks like this:

```
code  Swiss-Prot entry name(s)
101M  MYG_PHYCA   (P02185)
102L  LYS_BPT4    (P00720)
102M  MYG_PHYCA   (P02185)
103L  LYS_BPT4    (P00720)
103M  MYG_PHYCA   (P02185)
104L  LYS_BPT4    (P00720)
104M  MYG_PHYCA   (P02185)
105M  MYG_PHYCA   (P02185)
106M  MYG_PHYCA   (P02185)
```

5.4.3 search.pl

Here is a program that will accomplish the task oulined obove. Use `sprot-subset2.dat`, whick is a subset of the full Swissprot database. `sprot-subset2.dat` is available at the course homepage.

search.pl

```
1  #!/usr/bin/perl -w
2  ##### Program description #######
3  #
4  # Title: search.pl
5  #
6  # Author(s): Mattias Ohlsson
7  #
8  # Description:
9  #
10 # List of subroutines:
11 #
12 # Overall procedure:
13 #
14 # Usage: ./search.pl {table-file} {key-file} {swissprot-file}
15 #
16 ##################################
17 use strict;
18
19 #### The main program ####
20 unless ( @ARGV == 3 ) {
21     die "You must have three files as argument";
22 }
23
24 # Read the key file
25 my $keyref = readKey($ARGV[1]);
26
27 # Open the table that we should complete
28 open my $tableFH, '<', $ARGV[0] or die "Cannot open table file $ARGV[0]";
```

```
while ( my $line = <$tableFH> ) {
    next if ( $line =~ /^PDB/ );
    chomp $line;
    my $pdbID = $line;
    $pdbID =~ s/\s*$//;
    my $swissID = $keyref->{$pdbID};
    my ($os, $seq) = searchSwiss($swissID, $ARGV[2]);

    print "$pdbID\t$os\n";
}
close $tableFH;

#### End of main program #####


sub readKey {
    my ($keyFile) = @_;

    open my $kFH, '<', $keyFile or die "Cannot open file $keyFile";

    my %ptos;
    while ( my $line = <$kFH> ) {
        next if ( $line =~ /^code/ );

        # Each line contains 3 items, we are only interested in the first two
        my ($pdb, $sid) = split ' ', $line;

        # Store in a hash
        $ptos{$pdb} = $sid;
    }
    close $kFH;

    return \%ptos;

} # End of readKey


sub searchSwiss {
    my ($swissID, $swissFile) = @_;

    open my $sFH, '<', $swissFile or die "Cannot open file $swissFile";

    my $os = '';
    my $seq = '';
    while ( my $line = <$sFH> ) {

        # Check for the correct ID line
        next unless ( $line =~ /^ID\s+$swissID\s+/ );

        # Now scan for OS and SQ
        my $read = 0;
        while ( my $line = <$sFH> ) {

            # Stop if we hit the last of entry code
            last if ( $line =~ m|^//| );
```

```
84
85              # Collect the OS line
86              if ( $line =~ /^OS\s+(.+)$/ ) {
87                  $os .= $1 . ' ';
88              }
89
90              # Collect the sequence
91              if( $read ) {
92                  $seq .= $line;
93              } elsif ( $line =~ /^SQ/ ) {
94                  $read = 1;
95              }
96
97          }
98          last;
99      }
100     close $sFH;
101
102     $seq =~ s/\n//g;
103     $seq =~ s/\s//g;
104
105     return ($os, $seq);
106
107 } # End of searchSwiss
```

search.pl

5.4.4 Follow-up tasks 5-3

The following files are needed for the below exercises:

`search.pl`, `PDB-table.txt`, `pdb_seqres.txt`, `mkindex.pl`, `pdb2sprot.txt` and `sprot-subset2.dat`.

1. Make sure that you understand this program and then write a program header for `search.pl`.

2. Modify the program so that all of the (optional) `RX` lines are returned and printed instead of the organism.

3. The file `pdb_seqres.txt`[6] contain all PDB entries in Fasta format. Use this file to extract the protein sequences for all PDB codes in the `PDB-table.txt` file. Compare each sequence with the corresponding sequence from the Swiss-Prot database. Print the difference in length together with the other information.

4. Optimization of the code. The subroutine `searchSwiss` always starts from the beginning of the Swiss-Prot flat file. For relatively small files this is an acceptable solution, but for really large files this approach can take very long time.

   The Perl functions `seek` and `tell` can be used to "move around" in a file handle. Using the `tell` function we can create an index that relates a certain ID to a position

[6] ftp://ftp.wwpdb.org/pub/pdb/derived_data/pdb_seqres.txt.gz

in the file. With the use of this index and the `seek` function we can jump directly to the position of a certain ID.

The Perl script `mkindex.pl` contains a subroutine that makes such an index. Use this index and modify the `searchSwiss` routine to make use of the `seek` function. This will significantly speed up the code.

5.5 Blast parsing

In this section we will look at a Perl program that parses the large amount of information that a Blast run produces. Below is (part of) such a result file (`1TEN_blastp.res`) produced by the blastp program, when blasting the protein 1TEN (PDB id).

```
BLASTP 2.2.30+
Reference: Stephen F. Altschul, Thomas L. Madden, Alejandro
A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and
David J. Lipman (1997), "Gapped BLAST and PSI-BLAST: a new
generation of protein database search programs", Nucleic
Acids Res. 25:3389-3402.


Reference for compositional score matrix adjustment: Stephen
F. Altschul, John C. Wootton, E. Michael Gertz, Richa
Agarwala, Aleksandr Morgulis, Alejandro A. Schaffer, and
Yi-Kuo Yu (2005) "Protein database searches using
compositionally adjusted substitution matrices", FEBS J.
272:5101-5109.


RID: 3CDJ5NH701R


Database: All non-redundant GenBank CDS
translations+PDB+SwissProt+PIR+PRF excluding environmental samples
from WGS projects
           49,886,901 sequences; 17,905,752,166 total letters
Query=
Length=90


                                                                  Score     E
Sequences producing significant alignments:                      (Bits)  Value

pdb|1TEN|A  Chain A, Structure Of A Fibronectin Type Iii Domai...    182   4e-57
dbj|BAG64930.1|  unnamed protein product [Homo sapiens]              191   2e-54
.
.
.
ref|XP_005498448.1|  PREDICTED: tenascin isoform X3 [Columba l...    158   3e-42
gb|KFQ87172.1|  Tenascin-R [Phoenicopterus ruber ruber]              158   3e-42

ALIGNMENTS
>pdb|1TEN|A Chain A, Structure Of A Fibronectin Type Iii Domain From Tenascin
Phased By Mad Analysis Of The Selenomethionyl Protein
Length=90

 Score =   182 bits (461),  Expect = 4e-57, Method: Compositional matrix adjust.
 Identities = 90/90 (100%), Positives = 90/90 (100%), Gaps = 0/90 (0%)

Query  1   RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
           RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG
Sbjct  1   RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
```

```
Query  61  NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90
           NLKPDTEYEVSLISRRGDMSSNPAKETFTT
Sbjct  61  NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90


>dbj|BAG64930.1| unnamed protein product [Homo sapiens]
Length=1080

 Score =   191 bits (486),  Expect = 2e-54, Method: Composition-based stats.
 Identities = 90/90 (100%), Positives = 90/90 (100%), Gaps = 0/90 (0%)

Query  1    RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
.
.
.
```

There are of course many ways we can try to "summarize" the information found in this file. The program presented below parses a blast result file (like `1TEN_blastp.res`) in the following way: Alignments with the number of identities between a user defined interval are extracted and (optionally) printed on the screen together with the first identification line and the actual number of identities (in %).

parse.pl

```
1  #!/usr/bin/perl -w
2  ####### Program description #######
3  #
4  # Title: parse.pl
5  #
6  # Author(s): Mattias Ohlsson
7  #
8  # Description:
9  #  This is a parser for result files produced by the Blastp program
10 #  for sequence alignment.
11 #
12 # List of subroutines:
13 #  GetSubjects();
14 #  GetHSPs();
15 #  AnalyzeHSP();
16 #
17 # Overall procedure:
18 #  This program loads a result file from a blastp run. The program does
19 #  not check that the loaded file is a valid blastp file, only that it
20 #  is a non-empty file. The file is stored as a single scalar. All the
21 #  analysis is performed on scalar and not on an array containing lines
22 #  of the result file. The first step is to split the result into
23 #  subjects, this is performed in the GetSubjects routine. The program
24 #  then makes a loop over all found subjects and for each subject all
25 #  the high scoring pairs are extracted (GetHSP). A second loop is over
26 #  all HSPs found and where each HSP is analyzed in the AnalyzeHSP
27 #  routine.  This routine extracts all the information needed in order
28 #  to select the ones with a percentage of identities within a specified
29 #  range. The selected ones are printed together with the alignment
30 #  (optional).
31 #
32 # Usage:
33 #  ./parse.pl {blast result file}
```

```
#
###################################
use strict;

#### The main program ####

### Part 1:  Read the input file and check that is non-zero
undef $/;
my $Blast = (<>);
unless ( length $Blast > 0 ) {
    die "Zero length input file\n";
}

### Part 2: Some hardcoded constants
my $Icutlow = 80;
my $Icuthigh = 85;
my $PrintAlign = 1;

### Part 3: The rest
my @subjects = GetSubjects($Blast);

my (@Idents, @Alns);
foreach my $subj ( @subjects ) {

    my ($sID, $rHSPs) = GetHSPs($subj); # All the high scoring pairs

    undef @Idents;
    undef @Alns;
    for (my $i = 0; $i < @{$rHSPs}; $i++) {
        my $rHPSres = AnalyzeHSP($rHSPs->[$i]);
        my $id = $rHPSres->{'Pid'};

        # Check the criteria
        if ( $id >= $Icutlow && $id <= $Icuthigh ) {
            push @Idents, $id;
            push @Alns, $rHPSres->{'Aln'};
        }
    }

    if ( @Idents ) {
        print "======= ID: $sID  =======\n";
        foreach (my $i = 0; $i < @Idents; $i++) {
            print "=> Identities: $Idents[$i]%\n";
            if ( $PrintAlign == 1 ) { print $Alns[$i], "\n"; }
        }
    }
}


sub GetSubjects {
    # The blast "file"
    my ($blast) = @_; # Note this is a copy of the supplied blast "file"!

    # Get rid of everything before the ALIGNMENTS line
    $blast =~ s/^.*ALIGNMENTS\n//s;
```

```
89
90      # and everthing after the database statement
91      $blast =~ s/\s\sDatabase:.*//s;
92
93      # Now split on the >xxx pattern but keep the >xxx itself
94      my @subjects = split /(?=>\w{2,3})/, $blast;
95
96      return @subjects;
97
98  } # End of GetSubjects
99
100
101 sub GetHSPs {
102     # One subject
103     my ($subject) = @_; # Note this is a copy of the supplied subject!
104
105     # Extract an ID
106     my $id;
107     if ( $subject =~ /^>(\w{2,3}\|.*?\|)/ ) {
108         $id = $1;
109     } else {
110         print "Warning: Subject does not start with a known format\n";
111         print $subject;
112         exit;
113     }
114
115     # Now get HSPs by dividing at the Score line
116     $subject =~ s/^.*?(?=Score)//s;
117     my @hsps =  split /(?=\sScore)/, $subject;
118
119     return($id, \@hsps);
120
121 } # End of GetHSPs
122
123
124 sub AnalyzeHSP {
125     # The HSP
126     my ($hsp) = @_; # Note this is a copy of the supplied hsp!
127
128     # This will store the results
129     my %HSPres;
130     #Identities = 90/90 (100%), Positives = 90/90 (100%), Gaps = 0/90 (0%)
131     # Find the score
132     if ( $hsp =~ /Identities\s+=\s+\d+\/\d+\s+\((\d+)%\)/ ) {
133         $HSPres{'Pid'} = $1;
134     }
135
136     # Get the Alignment
137     $hsp =~ s/^.*?(?=Query)//s;
138     $hsp =~ s/\n(?=\n)//g;
139     $HSPres{'Aln'} = $hsp;
140
141     # Now return a reference to the result hash
142     return \%HSPres;
143
```

```
144 } # End of AnalyzeHSP
```
parse.pl

A few details in this program is worth noting.

- The program treats the input file as single scalar (`$Blast`) including all line breaks. A convenient way to read an file into a single scalar is to modify the special variable `$/`, called the input record separator. This variable is usually set to `\n`, the new line character. By undefining this variable we get the whole input file as a single scalar (see line 41-42).
- Pattern matching. Normally in a regular expression the dot (`.`) matches any character, except the new line one (`\n`). However if we use the `s` modifier it matches the new line character also. This is used in the program several times (e.g. line 88).
- Positive lookahead assertion. Sometimes it useful to have regular expression that matches in a hypothetical way. Perl have four such constructs. The *positive lookahead assertion* is used in this program and it looks like `(?=PATTERN)`. You can see an example of it at line 138. The regexp `s/\n(?=\n)//g` looks for two consecutives `\n\n` characters, but when it finds two it will on only replace the first one with "nothing" since the second is a lookahead assertion only. As they put in the Learning Perl textbook: *The Engine works it all out for us by actually trying to match the hypothetical pattern, and then pretending that it didn't match (if it did).*

This program run on the `1TEN_blastp.res` file (e.g. `>> ./parse.pl 1TEN_blastp.res`) results in the following output:

Result of: ./parse.pl 1TEN_blastp.res

```
1  ======= ID: ref|XP_005022418.1|  =======
2  => Identities: 84%
3  Query  1    RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
4              +LDAPSQIE KDVTDTTALITW KPLAEI+GIELTYG KDVPGDRTTIDL+EDENQYSIG
5  Sbjct  772  KLDAPSQIEAKDVTDTTALITWSKPLAEIEGIELTYGPKDVPGDRTTIDLSEDENQYSIG  831
6  Query  61   NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90
7              NL+P TEYEV+LISRRGDM S+P KE F T
8  Sbjct  832  NLRPHTEYEVTLISRRGDMESDPVKEVFVT  861
9
10 ======= ID: ref|XP_006137248.1|  =======
11 => Identities: 84%
12 Query  1    RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
13             +LDAPSQIEV+DVTDTTALITWFKPLAEID +EL+YG KDVPGDRTTIDL+EDE+QYSIG
14 Sbjct  804  KLDAPSQIEVRDVTDTTALITWFKPLAEIDDMELSYGPKDVPGDRTTIDLSEDESQYSIG  863
15 Query  61   NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90
16             NLKP TEYEV+LISRRGDM+S+P KETF T
17 Sbjct  864  NLKPHTEYEVTLISRRGDMTSDPVKETFVT  893
18
19 ======= ID: ref|XP_005022417.1|  =======
20 => Identities: 84%
21 Query  1    RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
22             +LDAPSQIE KDVTDTTALITW KPLAEI+GIELTYG KDVPGDRTTIDL+EDENQYSIG
23 Sbjct  772  KLDAPSQIEAKDVTDTTALITWSKPLAEIEGIELTYGPKDVPGDRTTIDLSEDENQYSIG  831
24 Query  61   NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90
```

```
            NL+P TEYEV+LISRRGDM S+P KE F T
Sbjct  832  NLRPHTEYEVTLISRRGDMESDPVKEVFVT  861

======= ID: ref|XP_005022415.1|  =======
=> Identities: 84%
Query  1    RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
            +LDAPSQIE KDVTDTTALITW KPLAEI+GIELTYG KDVPGDRTTIDL+EDENQYSIG
Sbjct  772  KLDAPSQIEAKDVTDTTALITWSKPLAEIEGIELTYGPKDVPGDRTTIDLSEDENQYSIG  831
Query  61   NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90
            NL+P TEYEV+LISRRGDM S+P KE F T
Sbjct  832  NLRPHTEYEVTLISRRGDMESDPVKEVFVT  861

======= ID: ref|XP_005022416.1|  =======
=> Identities: 84%
Query  1    RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
            +LDAPSQIE KDVTDTTALITW KPLAEI+GIELTYG KDVPGDRTTIDL+EDENQYSIG
Sbjct  772  KLDAPSQIEAKDVTDTTALITWSKPLAEIEGIELTYGPKDVPGDRTTIDLSEDENQYSIG  831
Query  61   NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90
            NL+P TEYEV+LISRRGDM S+P KE F T
Sbjct  832  NLRPHTEYEVTLISRRGDMESDPVKEVFVT  861

======= ID: gb|EOA99266.1|  =======
=> Identities: 84%
Query  1    RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
            +LDAPSQIE KDVTDTTALITW KPLAEI+GIELTYG KDVPGDRTTIDL+EDENQYSIG
Sbjct  772  KLDAPSQIEAKDVTDTTALITWSKPLAEIEGIELTYGPKDVPGDRTTIDLSEDENQYSIG  831
Query  61   NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90
            NL+P TEYEV+LISRRGDM S+P KE F T
Sbjct  832  NLRPHTEYEVTLISRRGDMESDPVKEVFVT  861

======= ID: ref|XP_005022414.1|  =======
=> Identities: 84%
Query  1    RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
            +LDAPSQIE KDVTDTTALITW KPLAEI+GIELTYG KDVPGDRTTIDL+EDENQYSIG
Sbjct  772  KLDAPSQIEAKDVTDTTALITWSKPLAEIEGIELTYGPKDVPGDRTTIDLSEDENQYSIG  831
Query  61   NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90
            NL+P TEYEV+LISRRGDM S+P KE F T
Sbjct  832  NLRPHTEYEVTLISRRGDMESDPVKEVFVT  861

======= ID: gb|KFQ40359.1|  =======
=> Identities: 84%
Query  1    RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
            +LDAPSQIE KDVTDTTALITW KPLAEI+GIELTYG KDVPGDRTTIDL+EDENQYSIG
Sbjct  454  KLDAPSQIEAKDVTDTTALITWSKPLAEIEGIELTYGPKDVPGDRTTIDLSEDENQYSIG  513
Query  61   NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90
            NL+P TEYEV+LISRRGDM S+P KE F T
Sbjct  514  NLRPHTEYEVTLISRRGDMESDPMKEVFVT  543

======= ID: ref|XP_005498448.1|  =======
=> Identities: 83%
Query  1    RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
            +LDAPSQIE KDVTDTTALITW KPLA+I+GIELTYG KDVPGDRTTIDL+EDENQYSIG
Sbjct  741  KLDAPSQIEAKDVTDTTALITWSKPLADIEGIELTYGPKDVPGDRTTIDLSEDENQYSIG  800
Query  61   NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90
            NL+P TEYEV+LISRRGDM S+P KE F T
```

```
Sbjct  801  NLRPHTEYEVTLISRRGDMESDPMKEVFVT  830

======= ID: gb|KFQ87172.1|  =======
=> Identities: 84%
Query  1    RLDAPSQIEVKDVTDTTALITWFKPLAEIDGIELTYGIKDVPGDRTTIDLTEDENQYSIG  60
            +LDAPSQIE KDVTDTTALITW KPLAEI+GIELTYG KDVPGDRTTIDL+EDENQYSIG
Sbjct  456  KLDAPSQIEAKDVTDTTALITWSKPLAEIEGIELTYGPKDVPGDRTTIDLSEDENQYSIG  515
Query  61   NLKPDTEYEVSLISRRGDMSSNPAKETFTT  90
            NL+P TEYEV+LISRRGDM S+P KE F T
Sbjct  516  NLRPHTEYEVTLISRRGDMESDPMKEVFVT  545
```

Result of: ./parse.pl 1TEN_blastp.res

5.5.1 Follow-up tasks 5-4

Download the files:

`parse.pl` and `1TEN_blastp.res`.

and complete the following tasks:

1. Add code the subroutine `AnalyzeHSP` to also extract the number of gaps (if any) in the alignment and print this on the screen.
2. For each HSP that you select, find the longest sub-alignment that contains only identities and print it on the screen. As an example, the following alignment

   ```
   Query: 41 FAGKDLESIKGTAPFETHANRIVGFFSKIIGELPN 75
             FAGKDL+S+K TA F THA RIVGF S+I+  + N
   Sbjct: 1  FAGKDLDSLKNTASFATHAGRIVGFVSEIVALMGN 35
   ```

   has `FAGKDL` as the longest sub-alignment with identities.
3. For each alignment that you select, count the number of sub-alignments, with only identities, larger than a given number.

Task 3 is of the type: "to do if you have time".

5.6 Hand-in exercise 3

Select one of the following problems as the hand-in exercise for this week.

1. The program for converting between clustalw and phylip file formats (first section of this chapter) does not handle the *conservation line* used in the clustalw files. Write a Perl program (or add more functionality to an existing one) so that when you convert from phylip to clustalw format the *conservation line* is also created. This program should also recognize the input format, i.e. one should not have to specify the direction of conversion. Below is a definition of the *conservation line*:

Three characters are now used in the conservation line:

'*' Indicates positions which have a single, fully conserved residue.

':' Indicates that one of the following "strong" groups is fully conserved: STA, NEQK, NHQK, NDEQ, QHRK, MILV, MILF, HY, FYW

'.' Indicates that one of the following 'weaker' groups is fully conserved: CSA, ATV, SAG, STNK, STPA, SGND, SNDEQK, NDEQHK, NEQHRK, FVLIM, HFY

Summary: You should write a Perl program that converts clustalw files to phylip files and vice versa. When you run your program on `clustalw1.align/clustalw2.align` you should get exactly the `phylip1.align/phylip2.align` and the other way around.

2. Write a parser for the result files produced by the FASTA sequence alignment program. This parser should, for each alignment, print the percentage of identities and print the length of the longest sub-alignment with only identities. To identify each aligned sequence, extract the first group characters after `>>` (e.g. `TR:E7EVS8_HUMAN`. The FASTA result file that you should work with is called `1TEN_fasta.res` and is available at the course web page.

   Summary: Write a parser for the `1TEN_fasta.res` file. Your output should look like the sample output file `sample-1TEN_fasta.txt`.