Perl을 이용한 Multiple FASTA File의 조작

작성자: 정 해 영 hyjeong@kribb.re.kr (한국생명공학연구원 미생물유전체연구실)
최초 작성일: 2003-05-20
최근 수정일: 2003-05-20

본 문서는 Perl을 이용하여 multiple fasta format의 서열 파일을 다루는 방법을 소개하고 있습니다. 펄의 기본적인 문법은 어느 정도 익숙하지만 실제의 서열 파일을 조작하는 방법을 알고 싶은 사람을 대상으로 씌어진 것입니다. BioPerl을 이용하면 Seq 및 SeqIO 개체를 이용하여 더 복잡한 작업을 할 수 있지만, 뭔가 마술 상자 같은 것에 모든 것을 맡기는 것이 석연치 않다는 생각이 드는 분, 직접 손으로 짠 코드를 가지고 서열을 다루고 싶은 분에게 조금이나마 도움이 되었으면 합니다.

펄의 문법이 워낙 유연하다 보니 제가 제시한 방법 이외에도 더욱 효율적이고 아름다운(?) 코드가 존재할 수 있을 것입니다. 또한 개인적인 코딩 성향 때문에 (예를 들어 함수 인자에 괄호를 둘러치지 않거나 my $var 형태로 변수의 scope를 엄격하게 정의하는데 게으르다거나) 보통 권고되고 있는 표준 코딩 스타일에 맞지 않을 수도 있습니다. 하지만 이런 형식에 너무 개의치 마시고 나름대로의 철학이 담긴 코드를 작성해 보시기 바랍니다.

기본 전제: 입력 서열 파일은 표준 입력으로 공급하고 출력물은 표준 출력으로 나가니 파일로 리다이렉션한다. 본문에서 제시한 펄 스크립트를 test.pl의 이름으로 저장하였다면 다음과 같이 실행하면 된다.

```
$ perl test.pl < input-fasta-file > output-file
```

Mission 1: 특정한 서열만 표준 출력으로 빼내고 싶다.

먼저 가장 단순한 방법에 대해 알아보자. 라인 단위로 읽어들여서 먼저 헤더인지(>로 시작) 아닌지를 판단한다. 헤더라면 적절히 처리하여 출력하기를 원하는 서열에 해당하는지를 판단한다. 그 다음 실제 염기열에 해당하는 라인을 만난 경우 출력을 해야 하는 서열이면 내보내고, 그렇지 않으면 skip한다. 서열 라인을 만났을 때 출력을 해야 하는지의 여부는 $mode 변수에 지정한다.

```
#!/usr/bin/perl -w
# Exercise 1

$to_extract = "test.seq";  # 출력할 서열의 id
$mode = 0;
```

```
while (<>) {
    if ( /^>/ ) {
        if ( /$to_extract/ ) {    # here!
            $mode = 1;
            print $_;   # 우선 헤더를 그대로 내보낸다.
        } else {
            $mode = 0;
        }
    } else {
        print $_ if ( $mode );
    }
}
```

이 간단한 방법의 문제점은 무엇일까? 우선 헤더 부분의 처리 방식이다. # here!로 표시된 줄에 주목하라. 단순히 헤더 라인에 "test.seq"라는 문자열이 포함되어 있다면 출력하는 것으로 판단하므로, 만약 헤더가 다음과 같이 되어 있다면 원래는 출력을 원하던 것이 아니었지만 결과는 원하는 대로 되지 않을 것이다.

```
>test2.seq    derived from test.seq
```

따라서 헤더 라인에서 공백을 포함하지 않는 최초의 문자열을 취하여 진정한 id로 삼는 것이 중요하겠다. Split 함수를 적절히 사용하여 각자 구현해 보라. 그리고 본 문서를 따라 직접 실습을 하면서 끝까지 가다보면 가장 현명한 방법을 찾게 될 것이다. 다음과 같이 헤더 라인의 상황은 여러 가지가 가능하니 어떤 경우에도 무난히 작동하는 코드를 생각해 보라.

```
>seq_id  description (공백으로 분리; 공백이 하나 이상일 수도 있음)
>seq_id description (탭으로 분리)
>seq_id (seq_id 바로 뒤에는 탭 또는 공백이 있을 수 있고, descrption 없이 바로 다음줄로 연결)
```

Mission 2: 뽑아내야 할 서열이 여러 개라면?

만일 뽑아내야 할 서열이 단 두 개이고 이를 각각 $to_extract1과 $to_extract2 변수에 저장하였다면 다음과 같이 해결하면 될 것이다.

```
#!/usr/bin/perl -w
# Exercise 2

( $to_extract1, $to_extract2 ) = ("test.seq", "test2.seq");
$mode = 0;
```

```
while (<>) {
    if ( /^>/ ) {
            if (/$to_extract1/ || /$to_extract2/ ) {   # 여기!
                $mode = 1;
                print $_;
            } else {
                $mode = 0;
            }
    } else {
        print $_ if ( $mode );
    }
}
```

"여기"라고 표시된 줄을 보라. 뽑아내야 할 서열의 이름을 지정한 변수가 단 두 개뿐이므로 이렇게 if 문 내에서 or 조건으로 연결하면 되겠지만 만일 수십개라면 어떻게 할 것인가? 일일이 별도의 변수에 할당하는 것도 문제이지만 if (/$to_extract1/ || $to_extract2/ || /$to_extract3/ || ...) 하는 식으로 조건식을 줄줄이 연결할 수도 없다. 당연히 이제는 배열을 사용하여 좀 더 효과적인 작업을 하도록 구상할 때가 되었다.

다음은 가장 무식한(?) 배열의 초기화 방법이다.

```
@files_to_extract = ( "test.ab1", "test2.ab1", "test3.ab1" );
@files_to_extract = qw( test.ab1 test2.ab1 test3.ab1 test4.ab1 );
```

그러나 배열을 쓰면서까지 이렇게 하는 것은 현명하지 못하다. 작업 대상 서열이 많다면, 가장 일반적인 방법은 별도의 목록 파일을 만들어서 읽어들이는 것이다. 이 방법은 각자 해결해 보기로 하고, "here ducuments"를 사용하여 코드 내에 데이터를 삽입하는 방법을 공부해 보자. 다음의 예제는 소스 내에 작업 대상 서열의 id를 포함시킨 다음 @to_extract 배열에 저장하는 작업 까지만 보여주고 있다.

```
#!/usr/bin/perl -w
# Exercise 2

@to_extract = take_list();
$num = $#to_extract;    # @to_extract 배열의 마지막 인덱스를 알아내기
for $i ( 0 .. $num ) {
    print $to_extract[$i], "\n";
}

sub take_list {
    $list =<< "END_OF_DATA";    # $list = <<END_OF_DATA라고 해도 됨
test1.ab1
```

```
test2.ab1
test3.ab1
test4.ab1
test5.ab1
END_OF_DATA
    @fields = split /\n/, $list;
    return @fields;
}
```

take_list()라는 서브루틴을 만들어서 그 내부에서 작업 목록을 처리하고 있다. Here document는 원래 본 쉘에서 유래한 것으로 여러 줄을 한번에 인용 처리하는 기법이다. << 뒤에 오는 "END_OF_DATA"는 일종의 delimiter로서 target이라고도 하며 here document의 내부에는 당연히 존재해서는 안될 것이다. Target은 코드의 가독성을 높이기 위하여 대문자로 쓰는 것이 좋을 것이다. << 기호의 뒤에 오는 target을 따옴표로 인용하지 않을 경우에는 << 뒤에 공백이 없어야 하는 것에 주의하라. Here document의 끝을 알리는 target은 반드시 라인의 맨 앞에 공백 없이 나와야 하며 뒤에도 공백이 있으면 안된다. Here document 부분은 마우스로 cut & paste를 하여 간편하게 가져다 붙이면 된다. 또는 vi의 : 명령어 모드에서 r command를 이용하여 다른 파일의 것을 읽어 들일 수도 있겠다. 소스의 가독성을 높이기 위하여 here document에 들여 쓰기를 적용하고 싶다면 이를 다시 제거할 방안도 강구해야 한다. 이에 대한 자세한 사항은 Perl Cookbook을 참조하기 바란다.

조금 옆길로 새는 기분이 들긴 하지만, here document를 다음과 같이 처리하는 것도 가능하다. 펄의 문법은 참으로 유연하지 않은가? $list라는 변수를 설정할 필요도 없이 @fields = split /\n/, <<END_OF_DATA; 라고만 작성해도 된다.

```
sub take_list {

@fields = split /\n/, ( $list = << "END_OF_DATA" );
test1.ab1
test2.ab1
test3.ab1
test4.ab1
test5.ab1
END_OF_DATA

    return @fields;
}
```

이제 해쉬를 이용한 좀 더 현명한 방법을 생각해 보자. 어떠한 아이템의 존재 여부를 확인하고 싶을 때 해쉬는 매우 훌륭한 해결책을 제공해 준다. 작업 대상 서열의 목록이 @to_extract 배열에 일단 저장되어 있다고 가정하자. @to_extract의 각 원소를 key로 하고 value는 적당한 아무 값을 넣어서 해쉬 변수를 만든 다음, 입력 파일의 헤더로부터 id를 추

출하여 이를 key로 하는 해쉬가 존재했었는지를 exists 함수로 확인하면 된다. value가 “”인 경우 실제 저장되는 값은 undef이다.

```
#!/usr/bin/perl -w
# Exercise 4

@to_extract = ("test.seq", "test2.seq");
%seen = ();
foreach ( @to_extract ) {
    $seen{$_} = "";
}
$mode = 0;
while (<>) {
   if ( /^>([^\s]+)/ ) {     # 1)
          if ( exists $seen{$1} ) {  # 2)
              $mode = 1;
              print $_;
          } else {
              $mode = 0;
          }
   } else {
       print $_ if ( $mode );
   }
}
```

1)의 if 문 조건식을 보라. 헤더 라인으로부터 어떻게 서열 id를 추출하고 있는가? 라인 맨 앞에 꺾쇠(>)가 있고 공백을 포함하지 않는 문자들이 연속해 나오는 것을 찾도록 정규식을 사용하였다. 괄호를 둘러친 것은 매치된 것을 $1, $2... 라는 변수의 형태로 쓸 수 있게 해 주는 것이다. 이 방법 말고도 서열 id를 뽑아내는 방법은 많다. 앞에서도 기술했듯이 헤더 라인을 공백을 경계로 split하여 배열에 저장한 다음 첫 번째 원소를 취하여 꺾쇠를 털어버리는 것도 좋다. 드물기는 하겠지만 헤더 라인의 필드 구분자가 스페이스 대신 탭일 가능성도 있으므로 이를 대비하는 것도 좋겠다.

2)에서 비로소 해쉬 변수가 쓰였다. 현재 읽어들인 헤더에서 서열 id를 추출한 다음 이를 key로 하는 해쉬가 존재하는지를 exist 함수로 확인하는 것이다. 실제 어떤 값을 value로 하고 있는지는 별로 중요하지 않다. 코드의 앞부분을 보면 단순히 null character가 지정되었음을 알 수 있다($seen{$_} = "";).

만일 뽑아낼 서열의 목록을 별도의 파일에 저장해 둔 상태라면 프로그램에서 이를 읽어들이면서 배열을 통할 필요 없이 직접 해쉬에 기록해 버리면 된다.

서열 리스트를 here document로 소스 내에 삽입한 상태에서 배열을 쓰지 않고 직접 해쉬로 목록을 저장한 뒤 작업을 하는 예를 살펴보도록 하자.

```
#!/usr/bin/perl -w
# Exercise 5

%seen = ();
$mode = 0;
take_list();   # 특별히 리턴을 하지는 않고 해쉬 설정 작업만 한다.

while (<>) {
    if ( /^>([^\s]+)/ ) {
            if ( exists $seen{$1} ) {
                $mode = 1;
                print $_;
            } else {
                $mode = 0;
            }
    } else {
        print $_ if ( $mode );
    }
}


sub take_list {
    foreach ( split /\n/, <<END_OF_DATA ) {
test.seq
test2.seq
END_OF_DATA

        $seen{$_} = "";
    }
}
```

Mission 3: 각 서열의 길이를 알아내고 싶다.

지금까지의 예에서는 헤더에서 서열 id를 찾아내어 출력을 해야 되는 것인지 아닌지를 판단한 다음 출력을 해야 하는 경우라면 다음 줄에 이어지는 서열 라인을 아무런 조작 없이 그대로 내보내도록 하였다. 이번에는 각 서열의 id와 길이(bp)를 출력하는 것을 목표로 한다.

작업을 할 서열의 헤더를 만날 때마다 @seq라는 배열을 초기화하고, 각 서열 라인은 공백을 경계로 split하여 @seq에 push 함수로 붙여나간 다음, 새로운 서열 헤더를 만날 때 출력하면 된다. 실제 구현은 조금 복잡해진다. 출력 모드인 경우 서열의 id를 다음 서열의 헤더를 만날 때까지 보유하고 있어야 하기 때문이다.

배열의 크기(즉 서열의 크기)를 알아내려면 스칼라 콘텍스트로 다루면 된다. 다음의 모범 답안을 잘 분석해 보라.

```
#!/usr/bin/perl -w
# Exercise 6

%seen = ();
$mode = 0;
take_list();

while (<>) {
    if ( /^>([^\s]+)/ ) {
        if ( $mode ) {
            print "Seq ID: $seq_id\tLength: ", scalar @seq, " bp\n";
            $mode = 0;
        }
        if ( exists $seen{$1}) {
            $seq_id = $1;
            $mode = 1;
            @seq = ();
        } else {
            $mode = 0;
        }
    } else {
         if ( $mode ) {
              chomp $_;
              push @seq, ( split //, $_ );
         }
    }
}
print "Seq ID: $seq_id\tLength: ", scalar @seq, " bp\n" if ( $mode ); # 여기!

sub take_list {
    foreach ( split /\n/, <<END_OF_DATA ) {
test.seq
test2.seq
END_OF_DATA

        $seen{$_} = "";
    }
}
```

if 문이 다소 복잡하게 되었지만 겁먹을 필요는 없다! 라인 단위로 입력 파일을 읽어 들인 다음 가장 먼저 할 일은 헤더인지 아닌지를 구별하는 것이다. 헤더인 경우 서열의 id를 정규표현식을 사용하여 $1 변수의 형태로 뽑아내게 된다.

만일 헤더라면, 현 상태가 출력 모드인지 아닌지를 $mode 변수의 형태로 구별한다. 만약 출력 모드인 상태에서 헤더를 만난 것이라면 결과(서열의 id와 길이)를 출력하면 된다. 여기서 한가지 조심해야 할 것은 $1 변수에 저장된 id를 그냥 출력하면 안된다. $1은 새로 만난 헤더에서 추출한 id일 따름이며 실제 서열의 id는 이미 지나쳐 버린 헤더 라인에 존재하는 것이기 때문이다. 따라서 exists 문이 나오는 블록 내에서 $1의 값을 $seq_id라는 별도의 변수에 저장해 두는 것이다. if 문의 경계가 어디까지인지를 유념하여 실행해 보기 바란다.

while 문이 끝난 다음에 나온 print문(# 여기!)은 무엇 때문에 필요한 것일까? 만일 입력 서열 파일의 맨 마지막 서열이 출력 대상이라면 다음 번 헤더를 만나지 못하므로 계산 결과를 출력할 기회를 읽게 된다. 따라서 while 종료시의 $mode를 확인하여 작업 대상이었다면 결과물을 출력하게 해 주는 것이다.

이렇게 서열을 배열에 저장해 두면 특정 염기서열의 패턴의 탐색, %G+C의 계산 및 서열 내 특정 범위의 추출 등 좀더 복잡한 작업을 수행하는 것이 가능해진다.

Mission 4: 가장 긴 서열을 출력하라.

Fragment assembly 작업 결과물을 분석할 때 흔히 접할 수 있는 문제이다. Phrap이나 cap3와 같이 ace file을 별도로 만들 수 있는 상황이라면 직접 서열 파일을 열어보지 않고도 어느 서열이 가장 긴 지를 알아낼 수 있다. 만일 그렇지 않다면? 앞의 예제에서처럼 배열의 크기로서 각 서열의 길이를 알아내는 것은 가능하다. 각 서열의 길이를 저장하는 해쉬(key: 서열 id, value: 길이)를 만들어 두면 어느 서열이 가장 긴 지를 알아낼 수 있다. 그러나 이는 입력 파일을 순차적으로 전부 읽어버린 다음에나 가능하다. 가장 긴 서열의 id를 알아낸 다음 입력 서열 파일을 다시 한번 읽을 것인가?

또는 입력 파일 내에 있는 서열간에 비교 분석을 해야 할 일도 있을 것이며, 프로그램 내부에서 필요에 따라 임의의 서열을 무작위적으로 접근하여 작업을 해야 할 경우도 있을 것이다. 지금까지 살펴 본 코드들은 서열 파일을 순차적으로 읽어나가면서 작업을 하는 경우에만 어울리는 것이었다.

그리고 서열에 부가되는 정보는 단순히 id만 있는 것이 아니다. 헤더 라인에 표시되는 다른 description field도 필요할 때가 있고 서열의 크기도 때로는 중요한 정보이다. 이것들을 서열 id를 이용하여 접근할 수 있는 별개의 데이터 구조(변수?)에 저장할 수 있다면 매우 유용할 것이다. C를 배운 경험이 있다면 구조체를 사용하면 이러한 목적에 합당한 코드를 작성할 수 있다.

펄의 기본 데이터 타입은 스케일러, 배열 및 해쉬 뿐이다. 그러나 Perl 5 버전에 이르러서는 참조(reference)가 도입되면서 좀 더 복잡한 구조의 데이터를 다루는 것이 가능해졌다. 펄에서 참조란 배열이나 해쉬 전체(혹은 그 어떤 것이라도)를 가리키는 스케일러 변수를 말한다. 스케일러 변수이므로 다른 배열이나 해쉬의 값이 될 수 있다. 원론적으로 배열이나 해쉬는 스캐일러 값만을 취할 수 있지만, 참조를 사용하면 다른 배열이나 해쉬를 지칭하는 것이 된다.

이 문서에서 참조의 모든 것을 설명하자면 너무 방대해지고, 또한 필자 자신도 참조의 모든 것을 알고 있지도 않다:) man perlreftut 명령어를 실행하여 관련 매뉴얼을 숙독하는 것이 큰 도움이 될 것이다. 잡소리이기는 하지만, 펄에 관한 어떤 책보다도 좋은 문서는 펄 패키지에 포함된 풍부한 매뉴얼들이다! 어떤 매뉴얼이 있는지 알고 싶은가? man perl 해 보라.

그러면 이번 미션에서 필요로 하는 데이터의 구조를 구체적으로 그려보도록 하자. 하나의 서열에 부속되는 정보를 다음의 4가지로 단순화시켜 보자.

- 서열 id
- 설명(description) : 서열 헤더 라인에서 id 뒤에 나오는 모든 정보 (즉 id와는 공백으로 구별됨)
- 서열의 길이
- 염기서열

참조의 기본을 이해하고 있다고 가정하고 해결책을 알아보자. 서열에 관련된 모든 정보의 집합을 $myseq{'id'}라는 해쉬를 통해 접근하게 해 보자. 물론 해쉬의 값은 스케일러이어야 하므로 설명(스케일러), 길이(스케일러) 및 염기서열(배열)이라는 세 가지 변수를 한번에 가질 수는 없다. 대신 'DESC', 'LENGTH' 및 'SEQ'를 key로 하는 anonymous hash를 설정한다. 마지막 해쉬는 스케일러 값을 가리키는 것이 아니라 또다시 배열에 대한 참조임을 주목하라. 너무 복잡한가? 다음은 참조가 가리키는 각 해쉬 또는 배열에 대해서 값을 할당하는 방법을 보여주고 있다.

```
$myseq{'id'}->{'DESC'} = "blabla...";
$myseq{'id'}->{'LENGTH'} = 509;
@{myseq{'id'}->{'SEQ'}} = qw( A A G T G A A G T A A C G T A G C A T );
```

헷갈리는가? %mysql은 id를 key로 하는 평범한 해쉬 변수이다. 그러나 각 key에 따르는 value는 단순한 스케일러 값이 아닌 또다른 익명 해쉬들이다. 즉 'DESC', 'LENGTH' 및 'SEQ'를 key로 하는 해쉬이다. 그러나 $myseq{'id'}->{'SEQ'}는 염기 서열을 저장하는 (익명) 배열에 대한 참조이다. 이것이 지칭하는 배열(즉 referent)는 @{myseq{'id'}->{'SEQ'}}와 같이 표현하면 된다. 여기에서 보인 것은 refenent 배열에 직접 값을 넣는 예이다. 다음의 줄은 염기 서열을 원소로 하는 익명의 배열을 설정한 다음 이것의 참조를 $myseq{'id'}->{'SEQ'}에 할당하는 것이다. 어느 것이나 결과는 똑같다.

```
$myseq{'id'}->{'SEQ'} = [ qw( A A G T G A A G T A A C G T A G C A T ) ];
```

코드는 다음과 같이 작성하면 된다. 여러 개의 서열을 읽어들이는 과정 중에서 특별히 작업할 대상을 선택할 필요가 없이 전체에 대해 작업을 하면 되므로 $mode 변수 같은 것을 쓸 필요가 없다. $myseq{'id'}->{'LENGTH'}가 지칭하는 배열 중에서 원소의 수가 가장 많은 것을 찾는 것은 매우 간단하므로 여기에서는 단순히 서열 id, 설명, 서열의 길이 및 맨 앞의 50 bp를 출력하는 것으로 대신하였다.

```
#!/usr/bin/perl -w
# Exercise 7

%myseq = ();

while (<>) {
    chomp $_;
    if ( /^>/ ) {

        if ( exists $myseq{$id} ) { # (1)
            $myseq{$id}->{'LENGTH'} = scalar @{$myseq{$id}->{'SEQ'}};
        }

        $_ =~ s/\t/ /g;  # 만일 탭이 있다면 전부 스페이스로 치환
        @fields = split / /, $_;
        ( $id = shift @fields ) =~ s/^>//;
        $desc = join ' ', @fields;
        $myseq{$id}->{'DESC'} = $desc;
    } else {
        push @{$myseq{$id}->{'SEQ'}}, ( split //, $_ );
    }
}
$myseq{$id}->{'LENGTH'} = scalar @{$myseq{$id}->{'SEQ'}};  # (2)

foreach ( keys %myseq ) {
    print $_, " ", $myseq{$_}->{'DESC'}, " ", $myseq{$_}->{'LENGTH'}, "\n";
    for $i ( 0 .. 59 ) { print $myseq{$_}->{'SEQ'}->[$i] }
    print "\n";
}
```

의외로 단순하지 않은가? # (1)의 if-block은 약간의 설명이 필요하다. 염기서열을 배열에 넣는 작업은 특별히 입력이 종료되었음을 알려줄 필요가 없지만, 서열의 길이를 알아내려면 다음 번 서열의 헤더 라인을 만났을 때 비로소 가능하다. # (1)은 다시 헤더를 만나게 되었을 때 서열의 길이를 계산하여 $myseq{$id}->{'LENGTH'}에 저장해 주는 기능을 한다. $myseq{$id}가 존재하는지 확인하는 것은 최초의 헤더를 그냥 지나가도록 하기 위함이다. # (2)가 없다면 무슨 일이 벌어질까? 마지막 서열의 경우는 다음의 헤더 라인 같은 것을 만나지 않고 while loop를 나가게 되므로 서열의 길이를 계산하지 않게 되므로 이를 방지하기 위한 코드인 것이다.

Mission 5: 레코드 형태로 서열 데이터를 다루어 보자!

C의 구조체나 파스칼의 레코드 형태로 서열 데이터를 다루어 보는 방법을 알아보자. 사실상 Mission 4의 정답에 모든 해결책이 다 들어있지만 레코드 자료형을 이용하여 좀 더 읽기 쉽고 세련된 코드를 만들어 보자.

펄에서는 참조를 사용하여 구조체를 구현할 수 있다. 다음의 코드 토막(code snippet)을 감상해 보자.

```
# Exercise 8

$record = {
    ID => "test1.seq",
    LENGTH => 15,
    SEQUENCE => [  qw( C A C T G A C G T G T A C G ) ]
   };
print $record->{ID}, "\n";
print $record->{LENGTH}, "\n";
print @{$record->{SEQUENCE}}, "\n";
```

$record 변수는 ID, LENGTH, 및 SEQUENCE를 key로 하는 익명 해쉬(anonymous hash)의 "참조"이다. $record->{SEQUENCE}가 가지는 값은 ID나 LENGTH와 같은 스케일러 값이 아니라 또다시 익명 배열에 대한 참조이므로 @{$record->{SEQENCE}}와 같이 표현하여야 전체 배열을 list context로서 다루게 된다. 만일 특정 원소만을 접근하려면 ${$record->{SEQUENCE}}[$i]처럼 복잡하게 쓸 것이 아니라 $record->{SEQUENCE}->[$i]라고 표기하면 된다. 위의 코드와 아래의 코드는 사실상 동일하지만 어느 것이 더 경제적인지 생각해 보라. $temp나 @seq_temp 같은 변수를 사용할 필요가 없는 것이다.

```
#Exercise 9

$temp{'ID'} = "test1.seq";
$temp{'LENGTH'} = 15;
@seq_temp = qw( C A C T G A C G T G T A C G );
@{$temp->{'SEQUENCE'}} = @seq_temp;
$record = \%temp;
print $record->{ID}, "\n";
print $record->{LENGTH}, "\n";
print @{$record->{SEQUENCE}}, "\n";
```

=> 표시를 사용하여 참조가 가리키는 익명 해쉬(즉 referent)의 key 및 value를 설정하는 경우 key를 인용 처리하지 않아도 되는 점에서 유리하다. 위의 예에서처럼 key를 대문자로 표현하면 가독성도 훨씬 좋아진다.

자, 그러면 지금까지 익힌 것을 이용하여 multiple fasta file 내의 서열을 레코드와 같은 데이터 구조로 읽어들이는 코드를 만들어 보자. Exercise 8과 9에서는 하나의 서열 데이터에 대해서만 작업하는 경우를 보이고 있지만 실제로는 훨씬 많은 데이터를 취급해야 한다.

```
#!/usr/bin/perl -w
# Exercise 10

%byName = ();

while (<>) {
    chomp $_;
    if ( /^>/ ) {

        if ( exists $byName{$id} ) { # (1)
            $byName{$id}->{LENGTH} = scalar @{$byName{$id}->{SEQ}};
        }

        $_ =~ s/\t/ /g;  # 만일 탭이 있다면 전부 스페이스로 치환
        @fields = split / /, $_;
        ( $id = shift @fields ) =~ s/^>//;
        $desc = join ' ', @fields;

        undef $record;    # (2)

        $record->{ID} = $id;
        $record->{DESC} = $desc;
        $byName{$id} = $record;
    } else {
        push @{$byName{$id}->{SEQ}}, ( split //, $_ );
    }
}
$byName{$id}->{LENGTH} = scalar @{$byName{$id}->{SEQ}};

foreach ( sort keys %byName ) {
    print $byName{$_}->{ID}, "\n";
    print $byName{$_}->{DESC}, "\n";
    print $byName{$_}->{LENGTH}, "\n";
    print @{$byName{$_}->{SEQ}}, "\n";
}
```

만들어 놓고 보니 사실상 Exercise 7과 크게 다를 바가 없다. # (2)를 눈여겨 보라.

$record가 구체적으로 어떤 해쉬 또는 배열의 참조인지를 보이지는 않았다. 대신 $id를 key로 하고 $record를 value로 하는 해쉬(%byName)를 만들었으며 다음번 while loop에서는 #(2)에서 $record를 undef 처리해 버렸다. 하지만 $record는 $byName{}의 값으로 살아있다! 만일 여기서 $record를 재정의하지 않으면 루프를 돌면서 계속 같은 referent에 데이터를 겹쳐 써 버릴 것이다.

Mission 6: 심화 학습을 원한다면...

더욱 공부의 수준을 높여나간다면 시퀀스를 객체로 처리하기 위한 self-made module 작성까지 나아갈 수 있을 것이다. 그러나 이 단계까지 왔다면 이제부터는 **BioPerl module**을 사용하여 작업하는 것이 훨씬 수월할 것이다.