課題 1-1（塩基使用頻度解析その 1）

0.1 課題 1-1

担当遺伝子配列をファイルから読み込み、その A,T,G,C 塩基の数をカウントするプログラムを書く。

先ずは課題をよく読んで、問題の意味、解法を考えてみて下さい。次に、プログラムの概略を考えましょう。

1 プログラムデザイン

1.1 概略

問題解決の方法を図式化するとプログラムを大体 3 つのパートに分ければ良いことがみえてきます。

1. データファイルを開く

2. データを読み込んで塩基を数える

3. 得られたデータを整理し、出力する

ここで、入出力の関数やファイルの取り扱いに関する知識が必要になりますが、これらは情処で受講した知識で十分対応できます。ただし、Perl では C や Java とは関数や文法が違いますので、「Perl 入門」（秀和システム）などを参照しながら課題を進めると良いでしょう。

問題解決の筋道を立てた後はそれを具体的にプログラミングしていけばよいのですが、ただ思いつくままにベタ書きしてもエラーも起こりやすくなります。より良い作業環境を目指して多少の準備をしておきましょう。

1.2 下準備

ここでいう下準備とは、そのプログラムでの一般的（グローバル）な設定をどうするかということです。Perl は自由なプログラミング言語なので、好きな形でいきなり書き出してかまいませんが、それまでにいろいろ考え ておいた方が便利なことがあります。少し考えてみましょう。

まず、目的となるデータを格納する変数を考えましょう。Perl では C や Java とは異なり、文字列でも数字でも、型を気にしないで変数に入れることができます。そして、Perl での変数は $var のように、$（ドルマーク）を先頭につけた文字列です。ここでは A,T,G,C の塩基の数を数えればいいわけですから､変数が 4 つ必要になります。更に塩基配列全体のデータを一時格納するために、変数を一つ用意してやれば便利でしょう。

これによりプログラムの核になるデータを扱う変数は 5 つとなります。

変数を定義する時には､なるべく my という変数を使うとよいでしょう。Perl では必ずしも変数を定義する必要はありませんが、その便利さゆえにプログラムがわかりにくくなってしまう場合があります。例えば while などのループの中に書いてしまうと、その関数内でのみ有効なローカルな設定となってしまうのでご注意下さい。今使っている変数がローカル変数なのかグローバル変数なのかは絶えず意識するようにして下さい。

最後に、保護モードが 755 の時に perl コマンドをうたなくても実行できるように、プログラムの先頭に

```
#!/usr/local/bin/perl
```

と書いておきましょう。こうすると、プログラムを実行する時に

```
% perl myprogram.pl
```

としなくても

```
% myprogram.pl
```

だけで実行できるようになります。

以上のことからプログラムの基本的な設定は以下のようになるでしょう。

```
#!/usr/local/bin/perl
my($seq, $A, $T, $G, $C);
```

ではいよいよプログラムの中身を考えていきましょう。今回はそれほど長くはならないみたいですのでサブルーチン化については考えないことにしておきます。（つまり全てそのまま書いてしまうということです。）

それではいきなりですが、データファイルを開いてみましょう。

2　ファイルオープン

2.1 ファイルハンドルについて

Perl は非常に協力なファイル入出力の仕組みを持っています。そして、Perl はプログラム内部でのファイルの扱いを簡単にするために、ファイルハンドルという変数とは違った方法でファイルを認識します。これは全て大文字で書かれる文字列（例：FILE）で、Perl のプログラム内部ではファイル名ではなくこのファイルハンドルを使います。

2.2 open

いよいよファイルを開きます。ファイルを開くには open という関数を使います。閉じるには close です。これはファイルに対する読み書き問わず共通です。

```
open(FILEHANDLE, "filename");
```

これがもっともプリミティブなファイルの開き方です。開くファイルを絶対パス名（カレントディレクトリにあれば省略）で特定します。これだけではファイルを読み取り専用で開きますが、書き込みや追加をしたい場合、それぞれファイル名の前に「>」や「>>」を追加します。例えば、”out.txt ”というファイルに出力したい場合、

```
open(OUT, ‘ >out.txt ’ );
```

とします。

比喩的に言うとファイルハンドルは任意のファイルに対する「パイプ」と言えます。役割に応じたパイプで使うファイルとプログラムをつないでやるわけです。データのやりとりは必ずこのパイプを通してやりとりします。つまり1度openで指定した後はファイル名の替わりにファイルハンドルをつかって作業をするわけです。さらにもう 1 つ意識して欲しいのは、このパイプは「動く」ということです。実は読み書きの場合には、ファイルハンドルはファイル全体を指すのではなくファイルのある特定の部分を指すと考えた方が良いと思われます。開いた直後はファイルの冒頭にあり、読み書きを繰り返すたびに前進し、常に新しいエリアを指しているということを少し念頭において下さい。

2.3 open をもっとクールに

ところで実際に open する時にはもう一工夫するとユーザにもプログラマーにも計算機にもやさしいプログラムにできます。

```
open(FILE, “my.seq”) || die(“ERROR: file does not exist\n”);
print STDERR“open data file my.seq\n";
```

この書式は読み込みファイルを開く時の「お約束」として覚えて下さい。プログラムを訳すとopen はもし失敗したら『そんなファイルなんか見当たらないよ』といってもらって終了し、成功（失敗しなかったら）すれば、その旨知らせて作業を続ける、となります。STDERR は標準エラー出力の識別子です。

2.4 close

```
close(FILEHANDLE);  # ファイルを閉じます
```

ファイルを閉じるには上のようにします。処理が終わった段階でファイルは必ず閉じて下さい。また閉じた後の FILEHANDLE はまた別のファイルを開くのに再利用することもできます。繰り返しますが FILEHANDLE は何でもいいのですが、open で開いたときと同じ名前である必要があります。

3　データ解析

以下に少々ヒントを挙げておきます。データの形もシンプルなので、少しハンドアウトを参考にしつつプログラミングしてみて下さい。

3.1 while 文を使ってファイルを最後まで…

ファイルを開いたらその内容を読みとって求めるデータ（ここでは塩基の数）を作成します。ここではファイルからデータを読み込む関数とその実際について説明します。基本的に「読み」はある一定の範囲（これは関数の種類による）のデータをファイルから取得し次に進みます。この「進める」時の方法として、普通 while 文を使ってファイルの最後まで「読んでは進み読んでは進み」を繰り返してやるわけです。

```
while(<FILE>){

}
```

上の形がそのような命令になっています。<FILE> のように <> でファイルハンドルを囲んだものは、一行ずつファイルから読み込んで、内容がある限り 1 を返します。ファイルの最後まで到達するとこれが 0 になるため、while 文が終了します。

while 文の中では、読み込まれた一行の文字列は $_ という特殊変数に代入されます。ですから、while 文の中では基本的にこの $_ という変数を使って作業します。

さて、while 文が出てきました。この繰り返し文は結構強力で、しかもこの場合データファイルに対する処理が終わるまで、ずっとぐるぐる回っているわけです。したがって、各変数の初期値などはこれ以前に設定しておく必要があります。while 文に入る前にはある程度の緊張感をもって「指さし確認」を行うぐらいの気持ちで確認しておくとエラーも少なくなるでしょう。

3.2 ファイル内容の保管

実際に塩基数を数える前に、まずファイルの中の塩基配列を一つの変数として読み込んでしまいましょう。これは、最初に宣言しておいた $seq という変数に、ファイルから読み込んだ内容をつなげていれてしまえばいいでしょう。while(<FILE>) では一行ずつファイルを読み込みますから、一行の終わりである改行コード（ ” \n ”）を取り除いたものを次々に連結していきます。

改行コードの除去には置換関数 s/// を使います。s/// は文字列処理が得意な Perl 特有の少し変わった関数ですが、非常に強力です。ここでは、スラッシュ (“ / ”) 記号に囲まれた部分に置換前文字列と置換後文字列を入れます。そして、置換元の文字列を =~ で代入します。例えば、$seq の中の改行コードを ENTER に変換したい場合、

```
$seq =~ s/\n/ENTER/;
```

とします。しかし、このままでは最初に登場する改行コードしか置換されないので、最後に g スイッチをつけて

```
$seq =~ s/\n/ENTER/g;
```

としてやると、全ての改行コードが ENTER に置換されます。

さて、少し上級者向けの説明をしましょう。s/// 関数は正規表現を使うことが可能です。ですから、例えば「小文字アルファベット以外を消す」（NULL 文字に置換する）ということを

```
$seq =~ s/[^a-z]//g;
```

という式だけで表現できてしまいます。Perl において正規表現とは非常に強力な機能ですので、余力がある人は参考書などを見て勉強してみてください。

次に文字列の連結ですが、これには .= 演算子を使います。$seq1 に $seq2 をつなげたい場合、

```
$seq1 .= $seq2;
```

とします。

3.4 文字数カウント

さて、次に $seq に入った配列データの中から A,T,G,C の数を数えてやらなければなりません。これには一文字ずつとりだして、

```
if ($nucleotide eq ‘a’){    # 文字列なので == ではなく eq で比較
  $A ++;
}
```

のように C 言語風の書き方をすることも可能ですが、せっかく Perl を使うのですから、Perl 独特な書き方を使いましょう。

先ほど説明しました s/// 関数に似た関数で、一文字置換関数 tr/// という関数が Perl には存在します。これは s/// とは違い、一文字だけ置換します。そして、この関数は特殊で、置換した数を返します。

例えば、a を t に変換したい場合、以下の二つは同じことをします。

```
$seq =~ s/a/t/g;
$seq =~ tr/a/t/;
```

しかし、tr/// は置換した数を返すので、

```
$count = $seq =~ tr/a/t/;
```

としてやると、$count に $seq の中にある a を t に置換した数が入ります。これを利用して、

```
$count = $seq =~ tr/a/a/;
```

と同じ文字を代入してやれば、その文字の数を簡単にカウントすることができます。

3.5 骨組み

大体プログラムの骨組みは以下のようになるでしょう。

```
#!/usr/local/bin/perl

my($A,$T,$G,$C,$seq);

open(FILE, ********************);

while(<FILE>){
   *************************  # 改行コードとりのぞき
   *************************  #$seq に読み込んだ行を連結
}
close(FILE);

$A = ******************
$T = ******************
$G = ******************
$C = ******************
```

4　データ出力

さてデータはできたはずです。あとはそのデータを表示するだけ。print を使えば簡単ですね。

```
print “A=$A\n”;
```

これで課題 1-1 は終わりです。1-2 及び 1-3 はこれをベースにすれば簡単でしょう。頑張ってみて下さい。