2 Block 2: Modifizieren der Dateneingabe und Datenausgabe

2.1 Lernziele

Ziel dieses Blocks ist es zu verstehen, in welcher Form Computerprogramme in der Bioinformatik *Daten einlesen* und *ausgeben* können, und in welchen logischen Schritten innerhalb eines einfachen Programmablaufs sie mit `Perl` umzusetzen sind. Dazu werden an mehreren Programmbespielen folgende programmatischen Elemente und die entsprechende `Perl` *Syntax* erläutert

- Dateneingabe über die Kommandozeile `<STDIN>`;
- Umleiten der Datenausgabe in eine Datei mit `open(FILE, ">Datei")`
    - Verwendung von *Dateihandles* `FILE`
    - Überschreiben eines Datei-Inhalts mit >;
    - Anhängen von Inhalten an eine bestehende Datei mit >>;
    - Spezifizierung und Schreiben des FASTA Sequenz(standard)formats;
    - Formatierung von tabulator-separierten Ausgabe-Dateien mit `print "\t"`;
- Dateneingabe aus / Lesen einer Datei mit `open(FILE, "<Datei")`
    - Verwendung von Array-Variablen `@` zum Speichern von Datei-Inhalten;
    - Ausgabe von Datei-Inhalten auf dem Bildschirm;
- Programmaufruf mit Kommandozeilen-Argumenten `@ARGV`
    - Ausgabe von `@ARGV` Array-Elementen mit `$ARGV[0]`.

2.2 Praxis

In den ersten beiden Programmierbeispielen `Seq_Aus.pl` und `Name_Aus.pl` haben wir gelernt wie man den Wert einer skalaren Variablen auf dem Computerbildschirm ausgeben kann.

Viele Computerprogramme in der Bioinformatik weisen Elemente die folgende typischen Schritte der Datenverarbeitung abdecken:

(1) **Daten einlesen**;

(2) **Daten prozessieren / analysieren** und

(3) die Resultate dann wieder als **Daten ausgeben**.

Beispielhaft werden wir nun betrachten wie Schritt (1) und Schritt (3) programmatisch umzusetzen sind.

In den folgenden Beispielen werden wir deswegen lernen wie

- Daten an eine Variable über die Kommandozeile übergeben werden können (Schritt 1: Daten einlesen);
- Die Datenausgabe vom Bildschirm (siehe vorhergehender Block) in eine Datei umgeleitet werden kann (Schritt 3: Daten ausgeben);
- Die Dateneingabe durch Öffnen und Lesen des Inhalts einer Datei erfolgen kann (Schritt 1: Daten einlesen).

2.2.1 Dateneingabe über die Kommandozeile `<STDIN>`

Im Beispiel `Seq_Aus.pl` hatte die Variable `$DNA_Sequenz` einen festen, im Programmtext über den Zuweisungsbefehl

```
$DNA_Sequenz = 'ACGGTGCTGGACTGATTAGCGTATATAGC';
```

definierten Wert: die Zeichenkette `ACGGTGCTGGACTGATTAGCGTATATAGC`.

Über den eine modifizierten Zuweisungsbefehl

```
$DNA_Sequenz = <STDIN>;
```

können wir der Variablen eine beliebige Zeichenfolge über die Kommandozeile zuweisen. `<STDIN>` steht für *Standard Input*, -also *Standardeingabe*-, die als die Bildschirm-Kommandozeile definiert ist. Analog ist `<STDOUT>` die *Standardausgabe*, ebenfalls die Bildschirm-Kommandozeile.

Der `print` Befehl wird immer auf `<STDOUT>` geleitet.

Das Programmbeispiel `Seq_In_Com.pl` illustriert die Verwendung in Zeile 6:

```
 1  #!/usr/bin/perl -w
 2  # Dieses Program liest eine DNA Sequenz von der Befehlszeile, speichert sie in
    einer Variablen und sie gibt auf den Bildschirm aus.
 3  # Ausgabe der Aufforderung eine Sequenz einzugeben auf den Bildschirm
 4  print "\nBitte Sequenz eingeben und Zeilenvorschub druecken: ";
 5  # Die eingebene Sequenz wir der Variablen zugewiesen
 6  $DNA_Sequenz = <STDIN>;
 7  # <STDIN> steht fuer Standard input
 8  # Entfernen des Zeilenvorschub-Steuerzeichens aus dem Wert der Variablen
 9  chomp $DNA_Sequenz;
10  # Ausgabe der Sequenz auf den Bildschirm
11  print "Die eingebene Sequenz lautet: $DNA_Sequenz";
12  # Ausgabe einer Leerzeile auf den Bildschrim
13  print "\n";
14  # Beenden des Programms
15  exit;
```

Um deutlich zu machen, dass das Programm auf eine Eingabe von der Kommandozeile „wartet“ ist Zeile 4

```
print "\nBitte Sequenz eingeben und Zeilenvorschub druecken: ";
```

eingefügt. Das Programm „wartet“ solange mit dem Ablauf, bis der Zeilenvorschub betätigt wird. Zeile 9

```
chomp $DNA_Sequenz;
```

enthält einen neuen Befehl. Der `chomp` Befehl angewendet auf die Variable entfernt das Zeilenvorschubszeichen `\n` aus dem Wert der Variablen.

2.2.1.1 Übung

2.2.1.1.1 Aufgabe

- Führen sie das Programm `Seq_In_Com.pl` auf Ihrem Computer aus. Beobachten sie die Ausgabe und studieren sie den Programmtext. Testen sie wie sich die Ausgabe (beachten sie die ausgegebenen Leerzeilen) verändert wenn sie Zeile 9 durch Einfügen eines Kommentarzeichens # am Zeilenanfang abwandeln.
- Schreiben sie auf der Basis von `Seq_In_Com.pl` ein neues Programm `Res_In_Com.pl`. Das Programm soll einen Namen für ein Restriktionsenzym (z.B. `HindIII`) und die entsprechende Erkennungssequenz (`AAGCTT`) in je einer Variablen speichern, -durch Zuweisung über die Kommandozeile. Die Ausgabe auf die Kommandozeile soll folgendermassen aussehen:

```
Name des Restriktionsenzyms: HindIII
Erkennungssequenz fuer HindIII: AAGCTT
```

2.2.1.1.2 Lösung

Programmtext von `Res_In_Com.pl`:

```
#!/usr/bin/perl -w
# Dieses Program liest den Namen und die Erkennungssequenz fuer ein
Restriktionsenzym von der Befehlszeile,
# speichert sie in je einer Variablen und sie gibt beides auf den Bildschirm
aus.
# Aufforderung einen Namen einzugeben
print "Bitte Namen fuer Restriktionsenzym eingeben und Zeilenvorschub
druecken: ";
# Der eingebene Name wird der Variablen zugewiesen
$Res_Name = <STDIN>;
# <STDIN> steht fuer Standard input
# Entfernen des Zeilenvorschub-Steuerzeichens aus dem Wert der Variablen
chomp $Res_Name;
# Ausgabe der Aufforderung eine Erkennungssequenz einzugeben
print "Bitte Erkennungssequenz eingeben und Zeilenvorschub druecken: ";
# Die eingebene Sequenz wird der Variablen zugewiesen
$Res_Seq = <STDIN>;
# <STDIN> steht fuer Standard input
# Entfernen des Zeilenvorschub-Steuerzeichens aus dem Wert der Variablen
chomp $Res_Seq;
# Ausgabe auf dem Bildschirm
print "Name des Restriktionsenzyms: $Res_Name\n";
print "Erkennungssequenz fuer $Res_Name: $Res_Seq\n";
# Beenden des Programms
exit;
```

2.2.2 Datenausgabe in eine Datei `open(FILE, ">Datei")`

Häufig müssen Daten nicht nur auf dem Bildschirm *temporär* ausgegeben, sondern auch in Ergebnis-Dateien *permanent* gespeichert werden. Solche gespeicherten Daten erlauben den Zugriff durch andere Programme, zum Beispiel innerhalb von komplexen, *Mehr-Schritt-Arbeitsabläufen*, so genannten *workflows*. D.h. die Ausgabe eines Arbeitschrittes kann als Eingabe für einen nächsten verwendet werden. Zugriff auf Ergebnisdaten innerhalb einer *Multi-User-Umgebung* ist ein anderer Grund Daten in Dateien zu speichern.

Wie bereits besprochen findet die Ausgabe in `Perl` standardmässig auf die Kommandozeile des Betriebssystems, auf `<STDOUT>` statt. Wir haben das bereits im Kontext des `print` Befehls kennen gelernt. Wenn die Ausgabe in eine Datei erfolgen soll, spricht man davon sie, -von der Kommandozeile-, *umzuleiten.*

Das folgende Programmbeispiel `Seq_Aus_Dat.pl` illustriert die Umleitung der Ausgabe in eine Datei:

```
 1  #!/usr/bin/perl -w
 2  # Dieses Programm gibt den Inhalt von zwei Variablen (ein Sequenzname und
    eine DNA Sequenz)
 3  # in eine Datei im aktuellen Arbeitsverzeichnis aus, deren Namen der Benutzer
    ueber die
 4  # Kommandozeile bestimmt.
 5  # Definiert den Namen der Sequenz
 6  $DNA_Name = 'CloneX.fasta';
 7  # Definiert die eigentliche Sequenz
 8  $DNA_Sequenz = 'ACGGTGCTGGACTGATTAGCGTATATAGC';
 9  # Fuegt die Werte der beiden Variablen zusammen - Konkatenierung
10  $Datei_Inhalt = ">$DNA_Name\n$DNA_Sequenz\n";
11  # Aufforderung einen Namen fuer die Ausgabedatei ueber die Kommandozeile
    einzugeben
12  print "Geben sie einen Namen fuer die Ausgabe-Datei ein: ";
13  # Einlesen des Namens
14  $Aus_Datei_Name = <STDIN>;
15  # Entfernen des Zeilenvorschubs aus dem Wert von $Aus_Datei_Name
16  chomp $Aus_Datei_Name;
17  # Oeffnen der Datei im aktuellen Arbeitsverzeichnis zum Schreiben
18  open (FILE, ">$Aus_Datei_Name");
19  # Ausgabe des Wertes von $Datei_Inhalt in die Datei
20  print FILE $Datei_Inhalt;
21  # Schliessen der Datei
22  close FILE;
23
24  # Ausgabe des Inhalts, der in die Datei geschrieben wurde, auf dem Bildschirm
25  print "Ausgabe in Datei $Aus_Datei_Name in aktuellem Arbeitsverzeichnis:\n";
26  print $Datei_Inhalt;
    # Beenden des Programms
    exit;
```

Das Beispiel enthält mehrere neue Befehle und Programmelemente, die im Folgenden besprochen werden:

Zeile 10

```
$Datei_Inhalt = ">$DNA_Name\n$DNA_Sequenz\n";
```

ist ein Zuweisungsbefehl. Der Variablen `$Datei_Inhalt` wird eine zusammengesetzte Zeichenkette zugewiesen, die aus dem Zeichen >, dem Inhalt der Variablen `$DNA_Name`, einem Zeilenvorschub `\n`, dem Inhalt der Variablen `$DNA_Sequenz` und noch einem Zeilenvorschub `\n` besteht. In doppelte Hochkommata stehende Variablen werden *interpoliert*. Das bedeutet, der Befehl setzt nicht die Zeichenketten `$DNA_Name` und `$DNA_Sequenz` selbst ein sondern die Werte der entsprechenden Variablen, die in Zeile 6 und 8 definiert wurden. Die zusammengesetzte Variable wird im späteren Programmverlauf für eine Ausgabe in eine Datei und eine Ausgabe auf den Bildschirm verwendet.

Die Zeilen 17 bis 22

```
# Oeffnen der Datei im aktuellen Arbeitsverzeichnis zum Schreiben
open (FILE, ">$Aus_Datei_Name");
# Ausgabe des Wertes von $Datei_Inhalt in die Datei
print FILE $Datei_Inhalt;
# Schliessen der Datei
close FILE;
```

enthalten die Programmelemente, die die Umleitung der Ausgabe in eine Datei kontrollieren. In Zeile 18

```
open (FILE, ">$Aus_Datei_Name");
```

wird eine neue Datei im aktuellen Arbeitsverzeichnis mit dem Namen, der in Zeile 14

```
$Aus_Datei_Name = <STDIN>;
```

über die Kommandozeile festgelegt wurde, geöffnet. Der `open` Befehl hat zwei durch Komma getrennte, in Klammern stehende *Argumente*. Das erste ist die so genannte *Dateihandle*, die in diesem Fall `FILE` heisst. Dieses Argument erlaubt später die Ausgabe des `print` Befehls umzuleiten (siehe unten). Das zweite Argument, in unserem Fall `">$Aus_Datei_Name"` bestimmt den Namen der Datei, die geöffnet werden soll, in unserem Fall der Wert der Variablen `$Aus_Datei_Name`, die in Zeile 14

```
$Aus_Datei_Name = <STDIN>;
```

über eine Zuweisung von `<STDIN>` definiert wurde. Das Zeichen legt fest, dass die Datei zum Schreiben geöffnet wird. Der Befehl

```
open (FILE, "<$Aus_Datei_Name");
```

würde die gleiche Datei, die dann natürlich bereits vorhanden sein müsste, zum Lesen öffnen (siehe unten).

Zeile 20

```
print FILE $Datei_Inhalt;
```

enthält den `print` Befehl, der jetzt an die Dateihandle `FILE` umgeleitet, also nicht auf `<STDOUT>` ausgegeben wird. Die Dateihandle bewirkt, dass der Wert der Variablen `$Datei_Inhalt` in die mit `FILE` verbundene Ausgabedatei geschrieben wird.

Zeile 22

```
close FILE;
```

schliesst die Datei, die mit der Dateihandle `FILE` assoziiert ist.

Einschub: FASTA Sequenzformat

Die Ausgabe der Sequenz auf dem Bildschirm und in die Datei entspricht dem so genannten *FASTA Sequenzformat*, das folgendermassen definiert ist:

- Zeile 1 / die *FASTA Kopfzeile:* Das > Zeichen gefolgt von einer beliebigen Folge von Zeichen und einem Zeilenumbruch `\n`. Nur das erste Zeichen darf > sein!
- Zeile 2 – x / *FASTA Sequenz*: Eine beliebige Zeichenfolge (sinnvollerweise entweder der Einzelzeichen-Code für Nukleotid- oder Aminosäuresequenzen). Zeilenenden, die Länge ist beliebig, werden durch `\n` dargestellt.
- Beispiel (inklusive der nicht-sichtbaren `\n` Zeichen für die Zeilenumbrüche):

```
>Test.fasta; eine einfache Testsequenz\n
AATGCTTGCGGGCTATATGCCTTGACGATTTGGGCCCGAT\n
ACCACCAC\n
```

Diese einfache Sequenzdarstellung ist sehr verbreitet, insbesondere als Eingabe-Format für viele bioinformatische Anwendungen. Viele Anwendungen unterstützen auch die Verwendung des so genannten *Multiplen FASTA Formats*:

```
>Test_1.fasta; eine einfache Testsequenz\n
AATGCTTGCGGGCTATATGCCTTGACGATTTGGGCCCGAT\n
ACCACCAC\n
>Test_2.fasta; noch eine einfache Testsequenz\n
AATGCTTGCGGGCTATATGCCTTGACGATTTGGGCCCGAT\n
ACCACCAC\n
>Test_3.fasta; eine letzte, einfache Testsequenz\n
AATGCTTGCGGGCTATATGCCTTGACGATTTGGGCCCGAT\n
ACCACCAC\n
```

2.2.2.1 Übung

2.2.2.1.1 Aufgabe

- Führen sie das Programm `Seq_Aus_Dat.pl` auf Ihrem Computer aus. Beobachten sie die Ausgabe und studieren sie den Programmtext.
- Schreiben sie auf der Basis von `Seq_Aus_Dat.pl` ein neues Programm `Res_Aus_Dat.pl`. Das Programm soll Namen und die entsprechende Erkennungssequenz für drei Restriktionsenzyme in Variablen speichern. Die Ausgabe dieser Daten soll in eine Datei `Enzym.tab` umgeleitet werden. Die Ausgabe-Datei soll folgendermassen formatiert sein:

```
Restriktionsenzym - Erkennungssequenz:

[Enzym_Name 1] - [Erkennungssequenz 1]
[Enzym_Name 2] - [Erkennungssequenz 2]
[Enzym_Name 3] - [Erkennungssequenz 3]
```

  Also zum Beispiel:

```
Restriktionsenzym - Erkennungssequenz:

HindIII - AAGCTT
BamHI - GGATCC
EcoRI - GAATTC
```

  Verwenden sie obige Restriktionsenzymdaten für die Umsetzung von `Res_Aus_Dat.pl`. Neben der Ausgabe in die Datei soll eine analoge Bildschirm-Ausgabe erfolgen.
- Schlagen sie Veränderungen vor, die das Programm benutzerfreundlicher und interaktiver machen (z.B. Eingabe des Namens für die Ausgabe-Datei etc.).

2.2.2.1.2 Lösung 1

Programmtext von `Res_Aus_Dat.pl`:

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

#  Dieses  Program  liest  den  Namen  und  die  Erkennungssequenz  von  drei
Restriktionsenzymen
# von der Befehlszeile, speichert sie in  Variablen und sie gibt sie in eine
Datei
# "Enzym.tab" im aktuellen Arbeitsverzeichnis und auf dem Bildschirm aus.

# Eingabe des Namens fuer das erste Restriktionsenzym
print "Name fuer Restriktionsenzym 1 eingeben und Zeilenvorschub druecken: ";
$Res_Name_1 = <STDIN>;
chomp $Res_Name_1;
# Eingabe der Erkennungssequenz
print "Erkennungssequenz fuer Restriktionsenzym 1 eingeben und Zeilenvorschub
druecken: ";
$Res_Seq_1 = <STDIN>;
chomp $Res_Seq_1;
# Wiederholung fuer zwei weiter Enzyme
print "Name fuer Restriktionsenzym 2 eingeben und Zeilenvorschub druecken: ";
$Res_Name_2 = <STDIN>;
chomp $Res_Name_2;
print "Erkennungssequenz fuer Restriktionsenzym 2 eingeben und Zeilenvorschub
druecken: ";
$Res_Seq_2 = <STDIN>;
chomp $Res_Seq_2;
print "Name fuer Restriktionsenzym 3 eingeben und Zeilenvorschub druecken: ";
$Res_Name_3 = <STDIN>;
chomp $Res_Name_3;
print "Erkennungssequenz fuer Restriktionsenzym 3 eingeben und Zeilenvorschub
druecken: ";
```

```
$Res_Seq_3 = <STDIN>;
chomp $Res_Seq_3;

# Zeilenweise Ausgabe in die Datei "Enzym.tab"
open (FILE, ">>Enzym.tab");
# Ausgabe der Kopfzeile
print FILE "Restriktionsenzym\tErkennungssequenz:\n\n";
# Ausgabe der anderen Zeilen
print FILE "$Res_Name_1\t$Res_Seq_1\n";
print FILE "$Res_Name_2\t$Res_Seq_2\n";
print FILE "$Res_Name_3\t$Res_Seq_3\n";
close FILE;

# Ausgabe auf dem Bildschirm
print "Ausgabe in Datei Enzym.tab in aktuellem Arbeitsverzeichnis:\n\n";
print "Restriktionsenzym\tErkennungssequenz:\n\n";
print "$Res_Name_1\t$Res_Seq_1\n";
print "$Res_Name_2\t$Res_Seq_2\n";
print "$Res_Name_3\t$Res_Seq_3\n";
exit;
```

2.2.2.1.3 Lösung 2

Zeile 30 (bitte beachten: im Programmtext es tatsächlich nur EINE Zeile)

```
$Datei_Inhalt = "Restriktionsenzymen - Erkennungssequenz:\n\n$Res_Name_1 -
$Res_Seq_1\n$Res_Name_2 - $Res_Seq_2\n$Res_Name_3 - $Res_Seq_3\n";
```

des Programmtextes von `Res_Aus_Dat.pl` ist sehr umständlich, da viele Werte einzelner Variablen zusammengesetzt werden müssen. Eine elegantere Lösung finden sie in Zeilen 30 bis 37

```
open (FILE, ">>Enzym.tab");
# Ausgabe der Kopfzeile
print FILE "Restriktionsenzym - Erkennungssequenz:\n\n";
# Ausgabe der anderen Zeilen
print FILE "$Res_Name_1 - $Res_Seq_1\n";
print FILE "$Res_Name_2 - $Res_Seq_2\n";
print FILE "$Res_Name_3 - $Res_Seq_3\n";
close FILE;
```

im Programm `Res_Aus_Dat_alt.pl` (der veränderte Block ist gelb hervorgehoben):

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

#  Dieses  Program  liest  den  Namen  und  die  Erkennungssequenz  von  drei
Restriktionsenzymen
# von der Befehlszeile, speichert sie in  Variablen und sie gibt sie in eine
Datei
# "Enzym.tab" im aktuellen Arbeitsverzeichnis und auf dem Bildschirm aus.

# Eingabe des Namens fuer das erste Restriktionsenzym
print "Name fuer Restriktionsenzym 1 eingeben und Zeilenvorschub druecken: ";
$Res_Name_1 = <STDIN>;
chomp $Res_Name_1;
# Eingabe der Erkennungssequenz
print "Erkennungssequenz fuer Restriktionsenzym 1 eingeben und Zeilenvorschub
druecken: ";
$Res_Seq_1 = <STDIN>;
chomp $Res_Seq_1;
# Wiederholung fuer zwei weiter Enzyme
print "Name fuer Restriktionsenzym 2 eingeben und Zeilenvorschub druecken: ";
$Res_Name_2 = <STDIN>;
chomp $Res_Name_2;
print "Erkennungssequenz fuer Restriktionsenzym 2 eingeben und Zeilenvorschub
druecken: ";
$Res_Seq_2 = <STDIN>;
chomp $Res_Seq_2;
print "Name fuer Restriktionsenzym 3 eingeben und Zeilenvorschub druecken: ";
$Res_Name_3 = <STDIN>;
chomp $Res_Name_3;
print "Erkennungssequenz fuer Restriktionsenzym 3 eingeben und Zeilenvorschub
druecken: ";
$Res_Seq_3 = <STDIN>;
```

```
chomp $Res_Seq_3;

# Zeilenweise Ausgabe in die Datei "Enzym.tab"
open (FILE, ">>Enzym.tab");
# Ausgabe der Kopfzeile
print FILE "Restriktionsenzym - Erkennungssequenz:\n\n";
# Ausgabe der anderen Zeilen
print FILE "$Res_Name_1 - $Res_Seq_1\n";
print FILE "$Res_Name_2 - $Res_Seq_2\n";
print FILE "$Res_Name_3 - $Res_Seq_3\n";
close FILE;

# Ausgabe auf dem Bildschirm
print "Ausgabe in Datei Enzym.tab in aktuellem Arbeitsverzeichnis:\n\n";
print "Restriktionsenzym - Erkennungssequenz:\n\n";
print "$Res_Name_1 - $Res_Seq_1\n";
print "$Res_Name_2 - $Res_Seq_2\n";
print "$Res_Name_3 - $Res_Seq_3\n";
exit;
```

Mit dem Befehl `open (FILE, ">> Enzym.tab ");` kann eine Datei zum Schreiben geöffnet werden. Im Gegensatz zu > bewirkt >> allerdings, dass der Inhalt an den bereist bestehenden Inhalt der Datei angefügt wird und ihn nicht überschreibt. Bei jedem Aufruf des Programms wird jetzt Inhalt an die Datei `Enzym.tab` *angehängt*. Löschen sie die Datei deswegen vor jedem Aufruf.

2.2.2.1.4 Aufgabe

- Modifizieren sie `Res_Aus_Dat_alt.pl` in einem neuen Programm `Res_Aus_Dat_tab.pl` so, dass die zwei Spalten in der ausgegebene Tabelle (auf dem Bildschirm und in der Datei) statt durch `Leerzeichen-Bindestrich-Leerzeichen` durch *Tabulatoren* getrennt sind. Die zu verwendende Syntax für den `print` Befehl ist folgendermassen definiert:

  ```
  # Ausgabe von tabulator-getrennten Spalten in Datei
  print FILE "$Res_Name_1\t$Res_Seq_1\n";
  # Ausgabe von tabulator-getrennten Spalten auf Bildschirm
  print "$Res_Name_1\t$Res_Seq_1\n";
  # "\t" fügt analog zum Zeilenumbruch "\n" einen Tabulator ein
  ```

  Der `print "\t"` befehl funktioniert also genauso wie der `print "\n"` Befehl. Tabulator-separierte Ausgaben tauchen häufig bei der Verwendung bioinformatischer Programme auf, weil so unterschiedliche Spalten aus Dateien relative einfach eingelesen werden können (siehe unten).

2.2.2.1.5 Lösung

Programmtext von `Res_Aus_Dat_tab.pl`(die veränderten Zeilen sind gelb hervorgehoben):

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

#  Dieses Program liest den Namen und die Erkennungssequenz von drei
Restriktionsenzymen
# von der Befehlszeile, speichert sie in  Variablen und sie gibt sie in eine
Datei
# "Enzym.tab" im aktuellen Arbeitsverzeichnis und auf dem Bildschirm aus.

# Eingabe des Namens fuer das erste Restriktionsenzym
print "Name fuer Restriktionsenzym 1 eingeben und Zeilenvorschub druecken: ";
$Res_Name_1 = <STDIN>;
chomp $Res_Name_1;
# Eingabe der Erkennungssequenz
print "Erkennungssequenz fuer Restriktionsenzym 1 eingeben und Zeilenvorschub
druecken: ";
$Res_Seq_1 = <STDIN>;
chomp $Res_Seq_1;
# Wiederholung fuer zwei weiter Enzyme
print "Name fuer Restriktionsenzym 2 eingeben und Zeilenvorschub druecken: ";
$Res_Name_2 = <STDIN>;
chomp $Res_Name_2;
```

```
print "Erkennungssequenz fuer Restriktionsenzym 2 eingeben und Zeilenvorschub
druecken: ";
$Res_Seq_2 = <STDIN>;
chomp $Res_Seq_2;
print "Name fuer Restriktionsenzym 3 eingeben und Zeilenvorschub druecken: ";
File Edit Options Buffers Tools Help
$Res_Name_3 = <STDIN>;
chomp $Res_Name_3;
print "Erkennungssequenz fuer Restriktionsenzym 3 eingeben und Zeilenvorschub
druecken: ";
$Res_Seq_3 = <STDIN>;
chomp $Res_Seq_3;

# Zeilenweise Ausgabe in die Datei "Enzym.tab"
open (FILE, ">>Enzym.tab");
# Ausgabe der Kopfzeile
print FILE "Restriktionsenzym\tErkennungssequenz:\n\n";
# Ausgabe der anderen Zeilen
print FILE "$Res_Name_1\t$Res_Seq_1\n";
print FILE "$Res_Name_2\t$Res_Seq_2\n";
print FILE "$Res_Name_3\t$Res_Seq_3\n";
close FILE;

# Ausgabe auf dem Bildschirm
print "Ausgabe in Datei Enzym.tab in aktuellem Arbeitsverzeichnis:\n\n";
print "Restriktionsenzym\tErkennungssequenz:\n\n";
print "$Res_Name_1\t$Res_Seq_1\n";
print "$Res_Name_2\t$Res_Seq_2\n";
print "$Res_Name_3\t$Res_Seq_3\n";
exit;
```

2.2.3 Dateneingabe aus / Lesen einer Datei `open(FILE, "<Datei")`

Das Lesen von Datei-Inhalten als Eingabe ist ebenfalls eine häufige Problemstellung in der Programmierung. Im folgenden werden die grundlegenden, dafür benötigten Elemente der `Perl` Syntax anhand des Programmbeispiels `Seq_In_Dat.pl` besprochen (die Zeilen mit neuen Befehlen und Programmelementen sind gelb hervorgehoben):

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

# Programm-Name: Seq_In_Dat.pl
# Autor: picker
# Datum: 11/03

# Dieses Program liest den Inhalt der FASTA Sequenzdatei "Test.fasta" im
# aktuellen Arbeitsverzeichnis ein und gibt ihn auf den Bildschirm aus.

# Definition des Namens der Datei, die geoeffnet werden soll
$Datei_Name = 'Test.fasta';
print "Der Inhalt von $Datei_Name ist:\n\n";
# Offnen der Datei zum Lesen mit dem  "<" Operator
open (FILE, "<$Datei_Name");
# Zuweisung des Dateinhalts an die Array-Variable @Datei_Inhalt
@Datei_Inhalt = <FILE>;
close FILE;
# Ausgabe des Variablen-Werts auf den Bildschrim
print @Datei_Inhalt;
exit;
```

Das Programm liest den Inhalt der FASTA Sequenzdatei `Test.fasta` im aktuellen Arbeitsverzeichnis ein und gibt ihn auf den Bildschirm aus.

Mit dem Befehl in Zeile 10

```
open (FILE, "<$Datei_Name");
```

wird die Datei mit dem Namen `Test.fasta` (gespeichert als Wert der Variablen `$Datei_Name`) im zum Lesen geöffnet. Beachten sie die Verwendung des < anstelle des > Operators (siehe vorhergehende Beispiele).

In Zeile 12

```
@Datei_Inhalt = <FILE>;
```

wird auf den Inhalt der Datei mit den Eingabe-Operatoren <> zugegriffen. Der Inhalt wir der Variablen `@Datei_Inhalt` zugewiesen. Beachten sie die neue Schreibweise mit `@` anstelle `$` vor dem Variablennamen: `@Datei_Inhalt` ist vom Variablen-Typ *Array*, nicht wie die bisher verwendeten Variablen, die auf `$` begannen, vom Typ *String*. Später werden wir den Unterschied verschiedener Variablen-Typen genauer kennen lernen.

In Zeile 15

```
print @Datei_Inhalt;
```

wird schliesslich mit dem `print` Befehl der Inhalt der Array-Variablen `@Datei_Inhalt` auf den Bildschirm ausgegeben.

2.2.3.1 Übung

2.2.3.1.1 Aufgabe

- Führen sie das Programm `Seq_In_Dat.pl` auf Ihrem Computer aus. Beobachten sie die Ausgabe und studieren sie den Programmtext. Modifizieren sie `Seq_In_Dat.pl` in einem neuen Programm `Seq_In_Dat_string.pl` so, dass der Datei-Inhalt von `Test.fasta` nicht in eine Variable vom Typ *Array* sondern in eine Variable vom Typ *String* gespeichert und ausgegeben wird. Gehen sie folgendermassen vor:

  (1) Ersetzen sie Zeile 12

  ```
  @Datei_Inhalt = <FILE>;
  ```

  durch

  ```
  $Datei_Inhalt = <FILE>;
  ```

  (2) Ersetzen sie Zeile 15

  ```
  print @Datei_Inhalt;
  ```

  durch

  ```
  print $Datei_Inhalt;
  ```

  Beobachten sie die Veränderungen in der Ausgabe im Vergleich zu `Seq_In_Dat.pl` (Erklärung siehe unten).

2.2.3.1.2 Lösung

Programmtext von `Seq_In_Dat_string.pl` (die veränderten Zeilen sind gelb hervorgehoben):

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

# Dieses Program liest den Inhalt der FASTA Sequenzdatei "Test.fasta" im
# aktuellen Arbeitsverzeichnis ein und gibt ihn auf den Bildschirm aus.

# Definition des Namens der Datei, die geoeffnet werden soll
$Datei_Name = 'Test.fasta';
print "Der Inhalt von $Datei_Name ist:\n\n";
# Offnen der Datei zum Lesen mit dem  "<" Operator
open (FILE, "<$Datei_Name");
# Zuweisung des Dateinhalts an die Array-Variable @Datei_Inhalt
$Datei_Inhalt = <FILE>;
close FILE;
# Ausgabe des Variablen-Werts auf den Bildschrim
print $Datei_Inhalt;
```

```
exit;
```

Das Programm `Seq_In_Dat.pl` produziert folgende Bildschirm-Ausgabe:

```
Der Inhalt von Test.fasta ist:

>Test.fasta; eine einfache Testsequenz in FASTA Format
AATGCTTGCGGGCTATATGCCTTGACGATTTGGGCCCGAT
ACCACCAC
```

Im Vergleich dazu produziert `Seq_In_Dat_string.pl` folgende Bildschirm-Ausgabe:

```
Der Inhalt von Test.fasta ist:

>Test.fasta; eine einfache Testsequenz in FASTA Format
```

Der Unterschied in der Ausgabe (gelb hervorgehoben für `Seq_In_Dat.pl`) betrifft die zweite und dritte Zeile des Datei-Inhalts von `Test.fasta`, der nur von `Seq_In_Dat.pl` aber nicht von `Seq_In_Dat_string.pl` dargestellt wird.

Beim Lesen des Datei-Inhalts von `Test.fasta` in die *skalare* Variable `$Datei_Inhalt` in `Seq_In_Dat_string.pl`

```
$Datei_Inhalt = <FILE>;
```

wird nur der Inhalt einer Zeile, in diesem Fall der ersten / FASTA-Kopfzeile, gelesen. Das heisst der Datei-Inhalt wird quasi als einzelne, zusammengesetzte Zeichenketten, die durch Zeilenumbrüche `\n` getrennt sind, gelesen. Mit einem zweiten und dritten Aufruf der Zeile

```
$Datei_Inhalt = <FILE>;
```

könnte also auch der Inhalt der zweiten und dritten Zeile von `Test.fasta` gelesen werden. Zur Darstellung des Gesamt-Inhalts ist das allerdings sehr umständlich.

Deswegen wird der Datei-Inhalt von `Test.fasta` im Programm `Seq_In_Dat.pl`

```
@Datei_Inhalt = <FILE>;
```

nicht in eine skalare Variable sondern in die Variable `@Datei_Inhalt` vom Typ *Array* gelesen. Ein Array ist eine Variable, die mehrere skalare Werte speichern kann, d.h. `@Datei_Inhalt` kann alle Zeilen von `Test.fasta` als eine Liste von skalaren Werten speichern. Ausgeschrieben hat die Variable `@Datei_Inhalt` nach der Zuweisung den Wert:

`(1.Zeile von Test.fasta, 2. Zeile von Test.fasta, ... , letzte Zeile von Test.fasta)`

In einem späteren Block werden wir genauer auf die Verwendung von Array-Variablen eingehen.

2.2.3.1.3 Aufgabe

- Schreiben sie anhand des in `Seq_In_Dat.pl` Gelernten ein neues Programm `Res_In_Dat.pl`. Das Programm soll den Inhalt von zwei existierenden Dateien in ihrem aktuellen Arbeitsverzeichnis

  `Test.fasta` – eine FASTA Sequenzdatei
  `Enzym_Master.tab` – eine Tabelle mit Restriktionsenzymdaten

  lesen und nacheinander auf dem Bildschirm ausgeben. Die Namen der beiden zu lesenden Eingabe-Dateien sollen dabei über eine Eingabe von `<STDIN>` *interaktiv* festgelegt werden.

2.2.3.1.4 Lösung

Programmtext von `Res_In_Dat.pl`:

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

```
# Programm-Name: Res_In_Dat.pl
# Autor: picker
# Datum: 11/03

# Dieses Programm liest den Inhalt einer FASTA Sequenzdatei und einer Datei
# mit Restriktionsenzymdaten im aktuellen Arbeitsverzeichnis ein und gibt
# beides nacheinander auf dem Bildschirm aus.

# Definition des Namens der FASTA Datei, die geoeffnet werden
# ueber die Kommando-Zeile
print "Geben sie den Datei-Namen fuer die Eingabe-Sequenz ein: ";
$Seq_Datei_Name = <STDIN>;
chomp $Seq_Datei_Name;

# Definition des Namens der Datei mit Restriktionsenzymdaten,
# die geoeffnet werden, ueber die Kommando-Zeile
print "Geben sie den Datei-Namen fuer die Restriktionsenzymdaten ein: ";
$Res_Datei_Name = <STDIN>;
chomp $Res_Datei_Name;

# Oeffnen, Lesen und Ausgabe der FASTA Datei
open (FILE, "<$Seq_Datei_Name");
@Seq_Datei_Inhalt = <FILE>;
print "\nDer Inhalt der Datei $Seq_Datei_Name ist:\n\n";
print @Seq_Datei_Inhalt;
close FILE;

# Oeffnen, Lesen und Ausgabe der FASTA Datei
open (FILE, "<$Res_Datei_Name");
@Res_Datei_Inhalt = <FILE>;
print "\nDer Inhalt der Datei $Res_Datei_Name ist:\n\n";
print @Res_Datei_Inhalt;
close FILE;
exit;
```

Das Programm produziert folgende Ausgabe auf dem Bildschirm:

```
Geben sie den Datei-Namen fuer die Eingabe-Sequenz ein: Test.fasta
Geben sie den Datei-Namen fuer die Restriktionsenzymdaten ein:
Enzym_Master.tab

Der Inhalt der Datei Test.fasta ist:

>Test.fasta; eine einfache Testsequenz in FASTA Format
AATGCTTGCGGGCTATATGCCTTGACGATTTGGGCCCGAT
ACCACCAC

Der Inhalt der Datei Enzym_Master.tab ist:

* Quelldatei fuer Restriktionsenzymdaten
* WWW Quelle: www.neb.com/neb/products/res_enzymes/re_update_frame.html
* Datum: 11/03
* Tabellenformat: "Name-Tabulator-Erkennungssequenz"
* Name  Erkennungssequenz
ApaI    GGGCCC
BamHI   GGATCC
BstXI   CCANNNNNNTGG
DraI    TTTAAA
EcoRI   GAATTC
EcoRV   GATATC
HindIII AAGCTT
KpnI    GGTACC
NarI    GGCGCC
NcoI    CCATGG
NotI    GCGGCCGC
PstI    CTGCAG
PvuI    CGATCG
PvuII   CAGCTG
SacI    GAGCTC
SacII   CCGCGG
SalI    GTCGAC
SmaI    CCCGGG
XbaI    TCTAGA
```

2.2.4 Programmaufruf mit Kommandozeilen-Argumenten `@ARGV`

Im Programm `Res_In_Dat.pl` haben sie gelernt wie über eine Eingabe von `<STDIN>` die Namen von Eingabe-Dateien, - die Beispiele waren `Test.fasta` und `Enzym_Master.tab` aus dem aktuellen Arbeitsverzeichnis -, interaktiv definiert werden können.

Wir werden jetzt lernen, wie bereits beim Aufruf des Programms so genannte *Kommandozeilen-Argumente* übergeben werden können, so dass das Programm danach ohne weitere Benutzer-Interaktion ablaufen kann. Das hat den Vorteil, dass der Benutzer nur einmal, - vor dem Start -, Eingaben machen muss, aber nicht währen des gesamten, möglicherweise sehr zeitaufwendigen weiteren Programmablaufs.

Betrachten sie den Programmtext des folgenden Beispiels `Res_In_Dat_Args.pl` (die entscheidenden Programm-Elemente und Kommentare sind gelb hervorgehoben):

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

# Programm-Name: Res_In_Dat_Args.pl
# Autor: picker
# Datum: 11/03
# Aufruf: perl Res_In_Dat_Args.pl [Dateiname Sequenz] [Dateiname Enzyme]

# Dieses Programm liest den Inhalt einer FASTA Sequenzdatei und einer Datei
# mit Restriktionsenzymdaten im aktuellen Arbeitsverzeichnis ein und gibt
# beides nacheinander auf dem Bildschirm aus. Die Namen der zu lesenden
# Dateien werden ueber Kommandozeilen-Argumente vor dem Programmstart
# vom Benutzer festgelegt.

# Die Kommandozeilen-Argumente werden automatisch in der Array-Variablen
# "@ARGV" gespeichert.

# Zugriff auf das erste Kommandozeilen-Argument [Dateiname Sequenz].
# Der Wert steht als String-Variable $ARGV[0] an der Position "0"
# im Array "@ARGV".
$Seq_Datei_Name = $ARGV[0];

# Zugriff auf das zweite Kommandozeilen-Argument [Dateiname Enzyme].
# Der Wert steht als String-Variable $ARGV[1] an der Position "1"
# im Array "@ARGV".
$Res_Datei_Name = $ARGV[1];

# Oeffnen, Lesen und Ausgabe der FASTA Datei
open (FILE, "<$Seq_Datei_Name");
@Seq_Datei_Inhalt = <FILE>;
print "\nDer Inhalt der Datei $Seq_Datei_Name ist:\n\n";
print @Seq_Datei_Inhalt;
close FILE;

# Oeffnen, Lesen und Ausgabe der FASTA Datei
open (FILE, "<$Res_Datei_Name");
@Res_Datei_Inhalt = <FILE>;
print "\nDer Inhalt der Datei $Res_Datei_Name ist:\n\n";
print @Res_Datei_Inhalt;
close FILE;
exit;
```

Wenn eine `Perl` Programm mit Kommandozeilen-Argumenten aufgerufen wird, in dem Fall von `Res_In_Dat_Args.pl` ist der Aufruf

```
perl Res_In_Dat_Args.pl [Dateiname Sequenz] [Dateiname Enzyme]
```

also z.B.

```
perl Res_In_Dat_Args.pl Test.fasta Enzym_Master.tab
```

wird automatisch eine Array-Variable `@ARGV` bereitgestellt. Wenn `n` Kommandozeilen-Argumente übergeben werden, kann auf diese Werte mit `$ARGV[0]`, `$ARGV[1]` ... `$ARGV[n]` zugegriffen werden. In `Res_In_Dat_Args.pl` wird zweimal auf Werte in `@ARGV` zugegriffen:

```
$Seq_Datei_Name = $ARGV[0];
```

```
$Res_Datei_Name = $ARGV[1];
```

Jede dieser Werte `$ARGV[0]`, `$ARGV[1]` ... `$ARGV[n]` ist ein separates *String* Element im Array `@ARGV`.

In einem späteren Block werden wir ausführlicher auf die Übergabe von *Kommandozeilen-Argumenten* eingehen und das Programm noch benutzerfreundlicher gestalten.

2.2.4.1 Übung

2.2.4.1.1 Aufgabe

- Führen sie `Res_In_Dat_Args.pl` auf Ihrem Computer mit `Test.fasta` und `Enzym_Master.tab` als Kommandozeilen-Argumenten mit folgendem Befehl aus:

  ```
  perl Res_In_Dat_Args.pl Test.fasta Enzym_Master.tab
  ```

  Betrachten sie die Ausgabe und studieren sie den Programmtext.

- Testen sie folgende Benutzungs-Szenarien von `Res_In_Dat_Args.pl`:

  i. Rufen sie das Programm mit Namen von zwei anderen Dateien (z.B. zwei `Perl` Programmen), die sich ebenfalls im aktuellen Arbeitsverzeichnis befinden, auf.

  ii. Rufen sie das Programm mit fehlerhaften Datei-Namen oder Namen von Dateien, die nicht existieren, auf.

  Beschreiben sie ihre Beobachtungen.