

# Lab Report 02

## Perl File Processing

Submitted By:	Hassan R. Al-Saffar
Student ID:	10190069
Section:	L02
Instructor:	Prof. Jeremy
Submission Date:	2017-Sep-25

## INTRODUCTION

---

Learning about file processing and loop constructs in Perl.

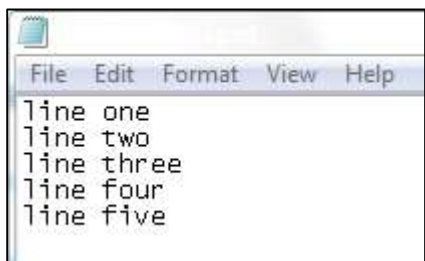
## ACTIVITIES

---

### ACTIVITY 1

For the purposes of this lab, it is necessary to create a source or test file to manipulated in file processing and different loop constructs.

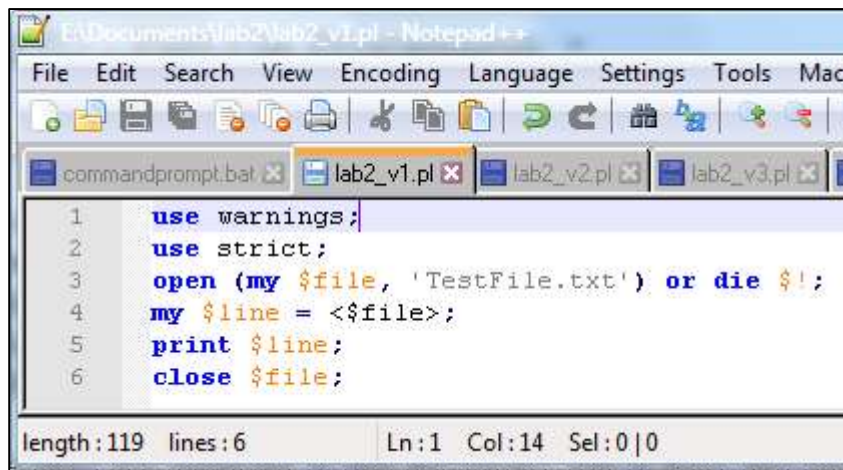
Using a text editor, I created a text file containing five lines.



*Figure 1: Text File presented in Notepad showing five lines in numerical order.*

### ACTIVITY 2

Printing a line from a scalar variable. I Created a Perl program that opens the file shown in *Figure 1*, I put the commands “use warnings;” and “use strict;”. This tells Perl to show error messages and tell me when quirks or minor issues in the program and to show in the command prompt. After this I show the program what file to open using “my \$file,” with the txt file name ‘TestFile.txt’. I then assigned <TEST\_INPUT>; to the variable \$line. In Notepad++ I put “my \$line = <\$file>; print \$line;” this tells Perl to print the first line, shown in *figure 2*.



```

1  use warnings;
2  use strict;
3  open (my $file, 'TestFile.txt') or die $!;
4  my $line = <$file>;
5  print $line;
6  close $file;

```

Figure 2: Perl file with an open my \$file or die directory, including a scalar variable printing the first line.



```

C:\> Command Prompt Portable
E:\Documents\lab2>lab2_v1.pl
line one

```

Figure 3: Version 1 shown output of figure 2, when ran in command prompt.

When *\$line* is a scalar variable only the first line is shown, because scalar values only show one line of the text file, in other words scalar variables show only one value. In this case I tell Perl to print the first line manually.

### ACTIVITY 3

In this Version, which I call version two, I have been tasked with printing from an array. In my program, I have changed the variable to an array by changing the *\$* to an *@*, so now *\$file* is now *@file*. From this activity, I learn the basic difference between scalars and arrays.

The following figure shows “lab2\_v2.pl” also known as version 2, tells Perl that “my @line = <\$file>;” which means that all lines in the document is assigned to the array, that is setup in this line.

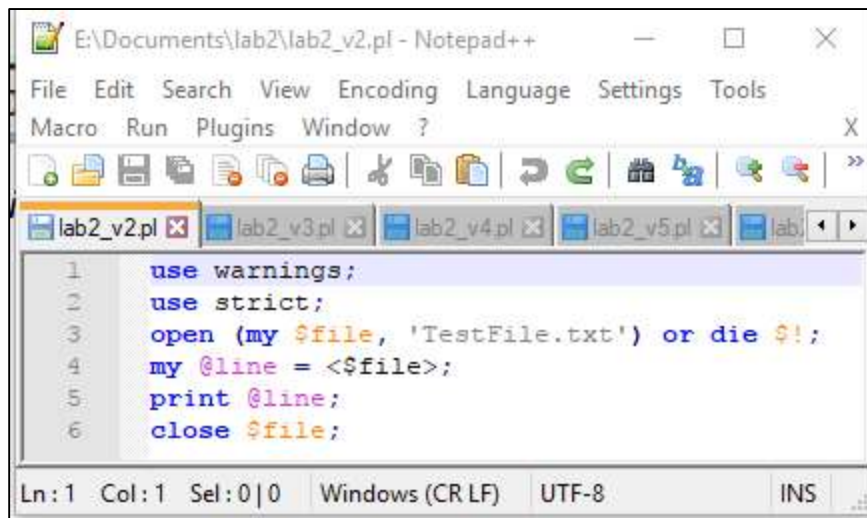


Figure 4: Notepad++ file showing array variable “@line” inputted to print ‘TestFile.txt’.

The reason that Perl decided to print all five lines, is due to the fact that arrays contain all of their strings in one value when commanded to output. Scalars only contain a single line of strings. The result of this caused Perl to print the text file in its entirety.

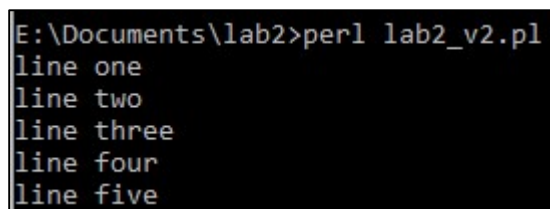


Figure 5: Result of Figure 4, lab2\_v2.pl ran on command prompt.

## ACTIVITY 4

In Version three, I try printing directly from the file handle. From the previous version I changed the variable back to a scalar. By doing this I reverted back to a version similar to the first version, the difference being the addition of one new line after the original print line. This new line will look like “print <\$file>” directly without first storing it in a variable.

Q: Can you explain what happened in these cases?

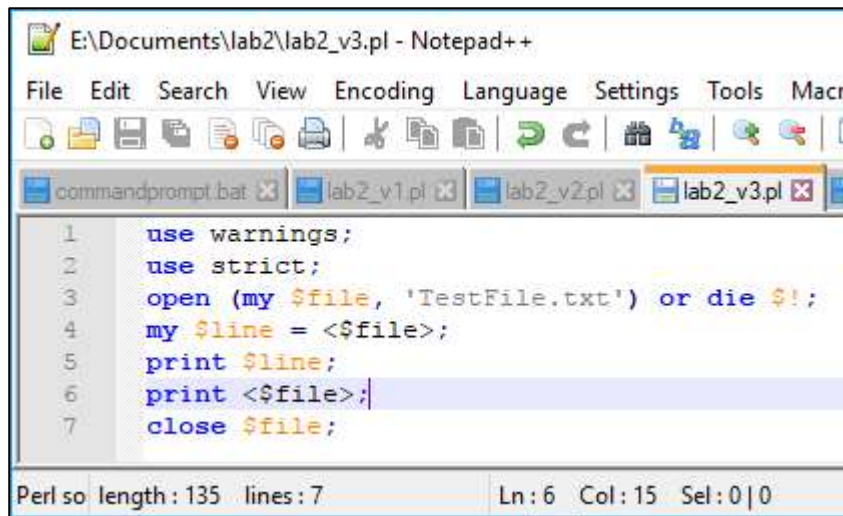
A: In this case Perl at line 5 in notepad++ followed the command “print \$line;” which tells it to print the first line in the txt document. In line 6 Perl reads the “print <\$file>;” script, because Perl remembers that it has printed line 1 already, it continues to print the rest of the remaining lines in the txt file.

Q: Was it what you expected?

A: My initial hypothesis was for Perl to show:

Line one  
Line one  
Line two  
Line three  
Line four  
Line five

I came to this conclusion because I assumed that Perl would not remember previous commands inputted and would simply print the contents of the file again.

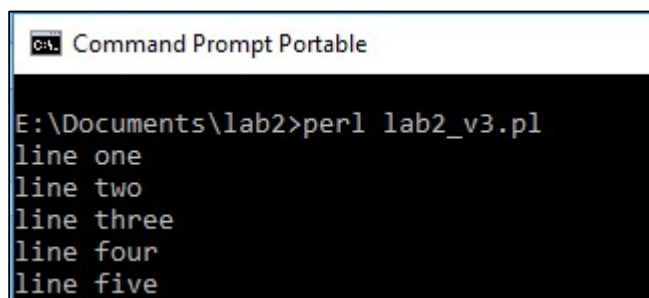


```

1  use warnings;
2  use strict;
3  open (my $file, 'TestFile.txt') or die $!;
4  my $line = <$file>;
5  print $line;
6  print <$file>;
7  close $file;

```

Figure 6: Notepad++ file of version 3 .pl file



```

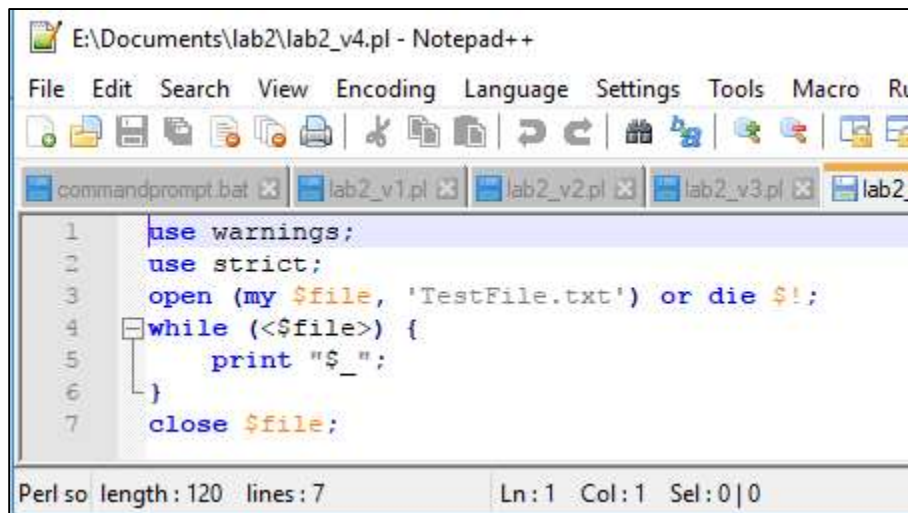
E:\Documents\lab2>perl lab2_v3.pl
line one
line one
line two
line three
line four
line five

```

Figure 7: Command prompt output of Figure 6

## ACTIVITY 5

In version four I created a program that loops through the file. I use the while loop command to tell Perl that it must repeat print each line in my txt document. Instead of using a “*print \$input\_line*” I instead used a “*\$\_*” scalar variable instead. The result of this is the same output when, I run the file shown in *Figure 9*.



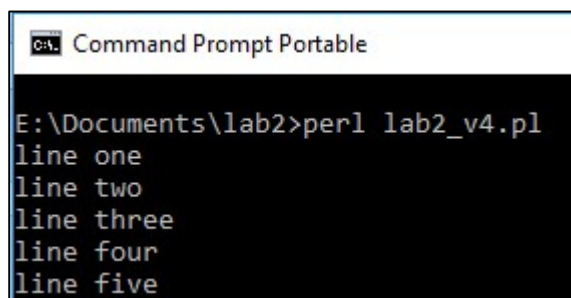
```

1  use warnings;
2  use strict;
3  open (my $file, 'TestFile.txt') or die $!;
4  while (<$file>) {
5      print "$_";
6  }
7  close $file;

```

Perl so length: 120 lines: 7 Ln: 1 Col: 1 Sel: 0 | 0

Figure 8: Version 4 .pl file with while loop command.



```

C:\> Command Prompt Portable

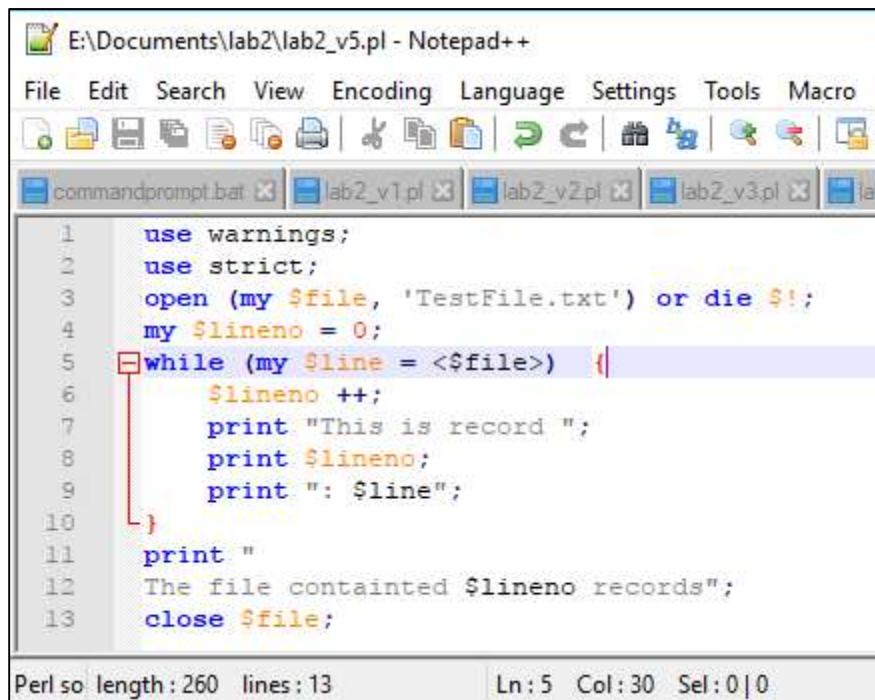
E:\Documents\lab2>perl lab2_v4.pl
line one
line two
line three
line four
line five

```

Figure 9: Output of Perl version 4 ran in command prompt.

## ACTIVITY 6

In the fifth version I explore counting records. I change the program so that it counts and labels the records as well as printing them. I Created a variable named “\$lineno” and initialized it to zero before the loop, and then I increment it on the first line inside the loop, I entered “\$lineno++” into the loop. This tells Perl in every lap in the loop to plus one the original numerical input, so zero would be 1 and so on. After the loop I printed a line saying, “The file contained \$lineno records.” where \$lineno is the record count. My next task was printing the records and totals. I did this by printing the line “This is record” & “print \$lineno” before the first line. The results are shown in Figure 10 and Figure 11.

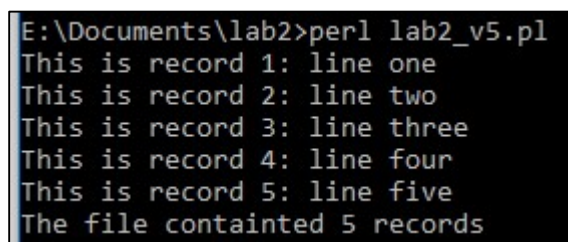


```

1  use warnings;
2  use strict;
3  open (my $file, 'TestFile.txt') or die $!;
4  my $lineno = 0;
5  while (my $line = <$file>) {
6      $lineno++;
7      print "This is record ";
8      print $lineno;
9      print ": $line";
10 }
11 print "
12 The file contained $lineno records";
13 close $file;

```

Figure 10: Version 5 input.



```

E:\Documents\lab2>perl lab2_v5.pl
This is record 1: line one
This is record 2: line two
This is record 3: line three
This is record 4: line four
This is record 5: line five
The file contained 5 records

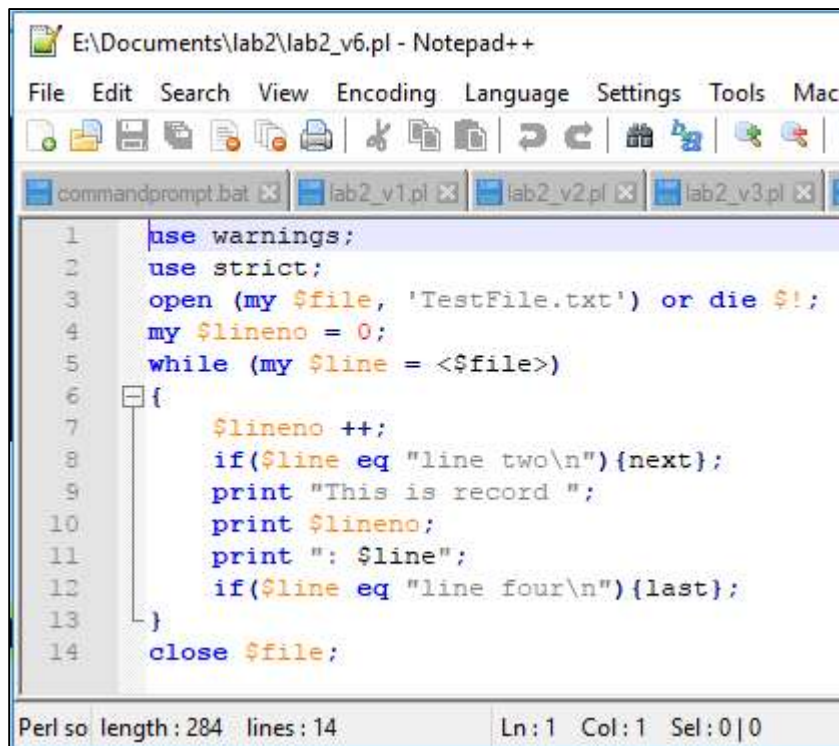
```

Figure 11: Version 5 output.

## ACTIVITY 7

Finally, we come to version six, in which I was tasked with manipulating Loop Control. I altered my program so it counted line two, but did not print it as well stopping after printing line four. To do this an if statement is needed to test the contents of the lines as they are read in each lap. In this version “next” and “last” is used to alter behavior of the loop. The next statement causes loop to return to the start, ignoring lines that come after. In our case this happens if the “if statement” is true. I also learned the last statement exits from the loop as if the “last” record is processed. To achieve this goal, we must keep in mind to put the “if true {next}” statement first and the “if true {last}” the last line in the loop. This is shown in figure 12 & figure 13.



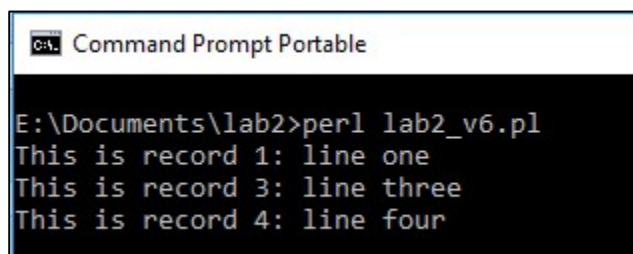


```

1 use warnings;
2 use strict;
3 open (my $file, 'TestFile.txt') or die $!;
4 my $lineno = 0;
5 while (my $line = <$file>)
6 {
7     $lineno++;
8     if($line eq "line two\n"){next};
9     print "This is record ";
10    print $lineno;
11    print ": $line";
12    if($line eq "line four\n"){last};
13 }
14 close $file;
  
```

Perl so length : 284 lines : 14 Ln: 1 Col: 1 Sel: 0 | 0

Figure 12: input of if true, next and last statements in Notepad++ Version 6.



```

E:\Documents\lab2>perl lab2_v6.pl
This is record 1: line one
This is record 3: line three
This is record 4: line four
  
```

Figure 13: Output of version 6 shown in figure 12.

In conclusion the use of next and last statements coupled with “if true” statement I was able to manipulate the loop process to change the results of the output. Perl is pretty cool dude.