About this tutorial

This tutorial on perl programming is written with curriculum of Anna University, Chennai for BE -CSE mind. The details are:

Anna University BE -CSE Curriculum R-2008- CODE CS2406 – OPEN SOURCE LAB

Para – 9 Text processing with perl.,simple programs,connecting with database e.g mysql

This tutorial is not aimed at text-book standard. It is to be treated as a quick guide for hands-on experience. Though the syllabus is single line, without understanding the basics of perl no text processing or database connectivity will be possible. Hence I have started from very basics – how to run perl onwards. I thank Raji Seetharaman for a reviewing and offering suggestions for improvement.

About perl
perl is simple and yet powerful scripting language. It can be used anywhere in the software eco-system – system administration, network, desktop or web. In this small notes we will learn how to write simple programs for practice in a lab.

Assumptions and limitations

a.It is assumed that perl is already installed in your system

b.It is assumed that you are running Linux

c.Error checking, debugging, best practices are not covered here

d. Anything given in bold+italics is to be practised by the student.

Structure of perl program

Perl program is a text file. You can use any text editor to create the program. Normally following line will be the first line

#! /usr/bin/perl

This tells Linux to use /usr/bin/perl executable to interpret rest of the lines in the program. This line may vary depending on the location of perl binary. Sometimes it may be /usr/local/bin/perl or some other place.

Commonly .pl extension is used, however you can write without extension also.

Starting with customary hello world program.

Use any editor and create a file with following contents

#! /usr/bin/perl

print “hello world\n”;

Assuming that you have saved it as prog.pl, we will see how to run the program.

Running perl program

Perl programs can be run in two ways. Assuming prog.pl is your file then

Method a:

$ perl prog.pl

Method b: Grant executable permission using chmod command and then invoking program name

$ chmod u+x prog.pl

$ ./prog.pl

or use full path name like

$ /home/ram/prog.pl

Note: For method b to work #! line is compulsory and ensure that #! occupies first and second character in the file.

If every thing goes well you will see hello world in your prompt.

Now that we know how to create and run let us go into language details.

1.Comments:

symbol is used for comments. All text from # till end of line is treated as comment.

e.g

 # This is a full line comment

print “hello”; # This is statement+comment

Note: There is no multiline comment.

2. Statement terminator ;

All Statements end with ; like C language.

3.print

print is simple function to display/output something on monitor/stdout. e.g

print “hello world”;

4.Variables

Variables are memory location to store information.

Variables are type less i.e there is no data type like int,char.

Every variable is a string and depending on the context will be treated as int, float etc.

There are 4 kinds of variables namely **scalars,lists,arrays,hashes.**

5 .Scalars

Scalar variables contain singular value like 10,hello etc

Name of scalar variable is prefixed with $ symbol.

eg.

$name=”ram”;# in string context

$age=30; # in numerical context

$age=$age+1; #treated as numeric

$age1=$age.$age; #treated as string. ‘.’(dot) is a concatenate operator

6.Handling quotes

” (double quote) is used when interpolation/substitution is required.

e.g

$name=”Raman”;

print “hello $name”;

will substitute $name with its value and output ‘hello Raman’.

‘ (single quote) is used when it is a literal string. Special characters will not be interpreted.

e.g

$name=’Raman’;

print ‘hello $name’;

The above line will print ‘hello $name’.

7.Lists

List variables are noted by symbol (). List is just a list of values – may be constants, scalars etc

e.g

(a,b,c) or ($name,$age,$sex)

They can be referred with index also. The index are specified inside a square bracket [ ].

e.g

$first=(a,b,c)[0];

print “$first\n”;

will output a.

List variables can be assigned like this

($name,$age)=(‘Raman’,20);

8.Conditionals – IF

The syntax of if statement is

if (condition) {

}

elsif (condition){

}

else {

}

The if statement is similar to if in C language, except

* flower brace is required even for single statement

* else if is noted by elsif (note missing e).

e.g

$mark=40;

if ($mark>75){

print “passed with distinction\n”;

}

elsif ($mark<35){

print “failed\n”;

}

else {

print “passed\n”;

}

Alternate form of *if* statement is

print “variable a is >10″ if ($a>10);

9.Accepting input

Keyboard inputs can be accepted using <STDIN>.

e.g

print “enter your name “;

$name=<STDIN>;

print “Welcome $name\n”;

Exercise:

Accept age. Type child if age below 12, type senior citizen when age above 60,otherwise type adult

10.Loops

for

for loop syntax is similar to c. It can also be used for iterating on a list. *foreach* is same as *for*. Both *for* and *foreach* are used interchangeably. for readability.

Classical *for* as in ‘C’

e.g.

for ($i=0;$i<10;$i++){

print “i=$i\n”;

}

The other way of using *for* is below.

foreach $i (a,b,c) {

print uc $i;

}

Explanation:

foreach will execute the body once for every element in the list – 3 times in this case. Each time the variable $i will get the value it is iterating ie. $i will be ‘a’ first time ‘b’ second time and ‘c’ the third time. uc – is a perl function to change a string into upper case.

You can combine functions like ‘print uc $i’ instead of print(uc($i)). Also note that brackets are optional for passing arguments to functions like uc, print.

The output will be ABC.

while

while loop is used to iterate and has syntax similar to C. e.g

$i=0;

while ($i<10){

print “i=$i\n”;

$i++;

}

11. Default scalar variable $_

$_ is called default variable. It will be used if no other variable is specified. We will see this by an example.

e.g

foreach (a,b,c){

print uc ;

}

The above foreach is similar to what is given under section10, however $i is omitted. Still perl will output same i.e ‘ABC’. This is because perl uses default variable $_ to store and expands the lines as

foreach $_ (a,b,c){

print uc $_;

}

Similarly $_ is used in the following case where ‘..’ the generator function is used.

foreach (1..10){

print ;

}

12.Arrays

Arrays are used to store multiple ordered values. Array variables should have prefix @. The size of array need not be specified beforehand. Each element of the array is scalar. Index starts with zero.

Whenever the whole array is required @array will be used. Suppose we want only an element then $array[ ] will be used as every element is a scalar, thus the prefix $.

e.g

@array=(1,2,3);

print @array;

Operations on Array

Assignment – whole array using list

@array=(1,2,3);

print "@array";

Assigning element

$array[3]=4;

print "@array \n";

New element can be appended at the end using *push* function

e.g.

push @array,'4′;

print "@array \n";

Last element can be removed using *pop* function

e.g.

$last=pop @array;

print "last=$last\n";

First element can be removed using *shift* function

e.g

@array=(1,2,3);

$first=shift @array;

print “first=$first\n”;

An element can be inserted at the beginning using *unshift*.

e.g

@array=(3,4,5);

unshift @array,’2′;

print “array=@array\n”;

Looping contents of an array using foreach

e.g

@array=(1,2,3);

foreach $i (@array){

print $i;

}

$#array is a special variable containing index of last element.It will be -1 for an empty array. In the above example *$#array* will be 2.

scalar(@array) is function to return the size of array.

Classical for can be used for iterating on array like this

e.g

@array=(1,2,3);

for ($i=0; $i<scalar(@array); $i++) {

print “i=$i array element=$array[$i]\n”;

}

for ($i=0; $i<=$#array; $i++) {

print “i=$i array element=$array[$i]\n”;

}

13.Hashes

Hash is associative/named array.They are key-value pairs. It is similar to array, except that we can use strings as index instead of 1..n. Hash variables will have **%** as prefix. The contents of hash are called values and index is called key.

e.g

%fruits= (

‘apple’ =>’red’,

‘banana’=>’yellow’,

‘grape’ =>’black’

);

Other way of populating a hash

e.g

%fruits =(‘apple’,'red’,'banana’,'yellow’,'grape’,'black’);

Here the list should contain even number of values. First element will be treated as key, second element value, third element key, fourth value and so on and so forth. In short odd elements will be keys, even elements will be values.

Individual elements of hash are accessed by means of *$hash{key}*

e.g.

print “colour of apple is $fruits{apple}\n”;

Adding new element

e.g.

$fruits{‘orange’}=’orange’;

Note the { } instead of [ ] as in the case of array;

Looping on hashes – keys function

e.g

foreach $f (keys %fruits) {

print “Color of $f is $fruits{$f}\n” ;

}

Explanation: keys is a function which return a list of key values.The list ( ) will contain apple,banana,grape while running.

14. Subroutines

Subroutines can be defined using sub keyword. The arguments passed will be in a default array @_;

e.g

$v1=10;$v2=20;

add($v1,$v2);

sub add {

($a,$b)=@_;

print $a+$b;

}

This should give output 30.

You can return value using return statement.

15. Scope of variables

By default all variables are global i.e available throughout the file. You can limit scope to a block/sub by using my.

e.g

```
$v1=10; $v2=30; #v1,v2 global

$v3=30;

$v3=add( $v1,$v2 );

sub add{

my ($i,$j)=@_;

print “inside add sub value of i=$i j=$j\n”;

print “inside add sub value of globals v1=$v1 v2=$v2 v3=$v3\n”;

return $i+$j;

}

print ” Value of globals v1=$v1 v2=$v3\n”;

print ” Value of scoped variables v3=$v3\n”;

print ” Value of variables inside sub i=$i j=$j\n”;
```

You can limit scope to a block also

e.g

```
for (my $i=0; $i<10; $i++ ) {

print “inside for i=$i\n”;

}

print “outside for i=$i\n”;
```

16.use strict

In perl you need not define variables before using. By default all variables are global. However, this may lead to errors due scope conflict or errors in naming. ‘*use strict*‘ is a pragma which will help in avoiding it. Once *use strict* is used, every variable has to be declared with proper scope using **my.**

e.g

```
use strict;
```

```
$v1=10;$v2=20;

add($v1,$v2);

sub add {

($a,$b)=@_;

print $a+$b;

}
```

The above code will not run and produce error. The corrected one will be like this

e.g

```
use strict;

my $v1=10;

my $v2=20;

add ( $v1,$v2 ) ;

sub add {

my ($a, $b)=@_;

print $a+$b;

}
```

17. References

References are address of the variable, similar (but not exactly) to pointers in c. You can take a reference by using **\.** It can be dereferenced by using **$$**;

e.g

```
$a=10;

$ref_toa=\$a;

print “value of a using reference = $$ref_toa\n Value of using directly=$a\n
Reference of a= $ref_toa”;
```

18.File handling

File handling can be done after opening a file and getting handle similar to C.

e.g

open($fh, “<”, “data.txt”);

here *$fh* – file handle which is a scalar variable. Also it is common to uppercased variable like FH.

< – open read only

data.txt – name of the file. Full path name should be given it it is not in current directory

File reading line by line can be done like

$line=<$fh>;

File writing

print $fh “hello”;

Example Problem: Open data.txt file. Copy contents to udata.txt duly converted into upper case

e.g

open ($fh, “<”, “data.txt”); #open file read only

open ($fh1,”>”,”udata.txt”); #Open file write mode

while ($line = <$fh>) { #read line by

print “line=$line”; #display content on screen

print $fh1 uc($line); #write upper cased content to new file

}

close($fh);

close($fh1);

19.Some common functions

uc :uc is used to convert string into all upper cases

e.g

$name='raman';

$name=uc ($name);

print "name=$name\n";

Similarly lc will convert into lowercase.

split: split is function which will split a string into components based on delimiter specified, and return a list of values.

eg. $line="20/12/2010";

($day,$month,$year)=split /\//,$line;

print "day=$day month=$month year=$year\n";

In the above example split will use / as delimiter. Note that / is special character. If we want literal / then we should use \/.

length: length returns the number of characters in a string.

20.Context

In perl many operations/functions perform differently based on context. For example @array will have different meaning like

@array=('a','b','c');

print "array=@array\n";

In the above line perl will print contents of the array.

$size=@array;

print "array = @array and size=$size\n";

In the above example perl will put 3 into $size i.e the number of elements in @array because it is used in scalar context (singular value).

Similarly split will return different values according to context. In the previous split returned a list of values. If used in scalar context split will return number of values i.e count

$line=”hello how are you”;

print “number of words=” . split / /,$line . “\n”;

21.Text Processing using perl.

Perl is powerful for text processing applications. For text processing Regular Expression (RE) helps a lot. Let us learn some basic Regular expressions.

Matching a pattern – m/ or just / is used to match a pattern in a string.

match a pattern syntax: /pattern/ or m/pattern/

This example uses default variable $_

$_=’hello how are you’;

if (/hello/){

print “default variable =$_\n”;

print “found hello\n”;

}

 # i modifier ignores case difference

if (m/HELLO/i){

print “found HELLo\n”;

}

#match with negation

if (! /hallo/){

print “not found hallo\n”;

}

print “\n\n”;

Bind Operator: In the above example we used default variable $_. You can use any variable for any RE using =~ which is called as bind operator.

```
#bind operator

#above example with a variable

$line='hello how are you';

print "line=$line\n";

if ($line =~ /hello/){

print "found hello\n";

}

print "\n\n";
```

Matching single character : . (dot)

dot matches any single character except new line.

at least one character should match

```
$line="help how are you";

print "line=$line\n";

if ($line=~ /hel.p/){

print "hel.p was found\n";

}

else {

print "hel.p was not found\n";

}

if ($line=~ /he.p/){

print "he.p was found\n";

}
```

else {

print “he.p was not found\n”;

}

print “\n\n”;

In the above example you will notice that first if condition fails. This is because there is no character after hel followed by p. The second if condition succeeds because in ‘help’ he is followed by a character and ‘p’.

Matching any single character zero or one time – ?

Question mark matches any character zero or one time in a string.

$line=”help how are you”;

print “line=$line\n”;

if ($line=~ /hel?p/){

print “hel? was found\n”;

}

else {

print “hel? was not found\n”;

}

#another example

$line=”help how are yu”;

print “line=$line\n”;

if ($line=~ /y?u?/){

print “you? was found\n”;

}

else {

print “you? was not found\n”;

}

print “\n\n”;

Matching a single character including newline – \s

\s is same as . but including \n

print ‘ \s is same as . but includes \n character’.”\n”;

$line= “hel\np how are you”;

print “line=$line\n”;

if ($line=~ /hel\sp/){

print ‘hel\s was found’.”\n”;

}

else {

print ‘ hel\s was not found’.”\n”;

}

print “\n\n”;

quantifier: zero or more times -

Quantifiers are symbols that will tell perl to match n number of times a particular pattern.

* star is a quantifier to match zero or more times

$line = “hello how are you”;

print “line=$line\n”;

if ($line=~ /hel*o/){

print ‘hel*o found’.”\n” ;

}

```
else {
print 'hel*o not found'."\n";
}
$line = "helo how are you";
print "line=$line\n";
if ($line=~ /hel*o/){
print 'helo found'."\n";
}
else {
print 'hel*o not found'."\n";
}
$line = "heo how are you";
print "line=$line\n";
if ($line=~ /hel*o/){
print 'hel*o found '."\n";
}
else {
print 'hel*o not found'."\n";
}
print "\n\n";
```

Match one or more time – + (plus)

plus quantifies one or more time;

```
$line = "hello how are you";
```

```
print “line=$line\n”;
if ($line=~ /hel+o/){
print ‘hel+o found ‘.”\n”;
}
else {
print ‘hel+o not found ‘.”\n”;
}
$line = “heo how are you”;
print “line=$line\n”;
if ($line=~ /hel+o/){
print ‘ hel+o found ‘.”\n”;
}
else {
print ‘hel+o not found ‘.”\n”;
}
print “\n\n”;
```

Grouping

You can group with ()

```
$line =”hhellohhelloh how are you”;
print “line=$line\n”;
if ($line=~ /h(hello)+h/){
print ‘(hello)+ found’.”\n”;
}
```

```
else {
print '(hello}+ not found '."\n";
}
$line ="hellohello how are you";
print "line=$line\n";
if ($line=~ /h(hello)*h/){
print '(hello)* found'."\n";
}
else {
print '(hello}* not found '."\n";
}
print "\n\n";
```

OR operator

| pipe is or operator used to connect to regex

```
$line ="vanakkam how are you";
print "line=$line\n";
if ($line=~ /(hello)|(vanakkam)/){
print '(hello)|(vanakkam) found '."\n";
}
else
{
print '(hello)|(vanakkam) not found '."\n";
}
```

print “\n\n”;

character class [0-9] [A-Z] [a-z]

[ ] : denotes character class e.g [A-Z] means A to Z

$line=”pc180a07″;

print “line=$line\n”;

if ($line=~ /pc[0-9]+[a-z]07/){

print ‘pc[0-9]+[a-z] occurred’.”\n”;

}

else {

print ‘pc[0-9]+[a-z] not occurred’.”\n”;

}

$line=”pc180b07″;

print “line=$line\n”;

if ($line=~ /pc[0-9]+[a,b]07/){

print ‘pc[0-9]+[a,b] occurred’.”\n”;

}

else {

print ‘pc[0-9]+[a,b] not occurred’.”\n”;

}

Inside a character class ^ negates

$line=”pcaa0b07″;

print “line=$line\n”;

if ($line=~ /pc[^0-9]+a/){

print 'pc[^0-9]+ found'."\n";

}

else {

print 'pc[^0-9]+ not found '."\n";

}

print "\n\n";

shortcut for [0-9] \d

\d : \d is a short cut for [0-9]

$line="pc180a07″;

print "line=$line\n";

if ($line=~ /pc\d+[a-z]07/){

print 'pc\d+[a-z]07 occurred'."\n";

}

else {

print 'pc\d+[a-z]07 not occurred'."\n";

}

tr operator translates characters in a list

$line="original string";

print "line=$line\n";

$line=~ tr/r/xx/;

print 'after applying tr/r/xx the result is '."\n"."$line\n";

print "\n";

$line="original string";

print “line=$line\n”;

$line=~ tr/rin/RIN/;

print ‘after applying tr/rin/RIN/ the result is ‘.”\n”.”$line\n”;

print “\n\n”;

Substitution

s/// substitutes a substring with given string

$line=”original string”;

print “line=$line\n”;

$line=~ s/r/xx/g;

print ‘after applying s/r/xx the result is ‘.”\n”.”$line\n”;

print “\n”;

Perl has more features than shown above as far as regex is concerned. With what we saw above let us do few exercises.

Exercise

1.Read a file. Count number of words in it. To keep it simple let us define word as a set characters followed by a space. Thus ‘hello how’ will be two words and ‘hello how .’ will be three words.

#! /usr/bin/perl

print “Enter File Name “;

$filename=<STDIN>;

#First open the file for reading

open ($fh,”r”,$filename);

#read line by line until end of file

while ($line=<$fh>){

@words=split $line; #break lines in words and store in an array.

$nw = $nw + scalar(@words); #no.of element in array = no of words.

}

print "Number of Words in the file : $nw \n";

close ($fh);

Practice : 1a. Modify above program to omit punctuation marks like ., etc and then count

1b. Modify above program without using scalar().

1c. Modify the above exercise to count distinct words

1d. Modify the exercise to count all distinct words whose length is 5 or more

1e. Modify the exercise to accept a word and then count occurence of that word in a file

2. A text file contains following data in a comma separated method

101,ram,manager,20000

102,baskar,ceo,40000

103,ravi,executive,12000

104,arun,executive,10000

It represents employee details in a firm viz. empid,name,designation,salary. Generate an output containing following details

Designation No.of Emp. Total Salary

ceo 1 40000

executive 2 22000

manager 1 20000

List of employees in alphabetical order

Name

arun

baskar

ram

ravi

```
#! /usr/bin/perl

use strict;

#declare variables

my ($fname,$fh);

my %desig_salary; #Hash to store summary of salary

my %desig_empno; #Hash to store number of employees per designation

my @names; #array to store names

my ($line,$empid,$name,$desig,$salary); #variables to store data from file

print “enter data file name “;

$fname=;

chomp($fname);

open( $fh,”<”,$fname) or die “could not open file “; #open file for reading

while ($line=<$fh>){ #read one line

($empid,$name,$desig,$salary)=split /,/,$line; #break line into values

#add 1 to no.of.employees for this designation

$desig_empno{$desig} = $desig_empno{$desig} + 1; #here explicit addition is used

#add salary to desig hash

$desig_salary{$desig} += $salary; # Here += style addition is used. Note += is similar to C

push @names,$name;

}

close($fh);
```

#desig_salary has contains a hash whose keys are designation and values are the sum of salary

#iterating on this hash will print first part of our problem

printf "%15s\t%15s\t%15s\n",'Designation','No.of.Emp','Total Salary';

foreach my $key (keys %desig_salary){

printf "%15s\t%15d\t%15f\n",$key,$desig_empno{$key},$desig_salary{$key};

}

#Second part printing names in alphabetical order

print "Name\n";

foreach $name(sort @names){

print "$name\n";

}

3.Accept a host name from the user. Scan through the hosts file and find out the IP number.

Ensure that comments are not processed. Normally the hosts will contain ipnumber and hostname one entry per line. Comments will start with # and end at the line. e.g

127.0.0.1 localhost

192.168.1.1 mygateway

192.168.1.2 myserver # This is my server

#hosts in another LAN

192.168.2.1 anothergateway

#! /usr/bin/perl

print "enter hostname ";

$hname=<STDIN>;

chomp $hname; #remove newline character

```
open($fh,'<','hosts');

while ($line=<$fh>){

#first remove any comments in the line by substituting with null

$line=~ s/(\#.*\n)//;

#print "line after=$line\n";

($no,$name)=split ' ',$line; #note split ' '. This is a special

#delimiter which takes care of multiple

#whitespaces.

#if hostname is found print the ipno and get out of while loop

if ($name eq $hname){

print "hostname=$name IP No.:=$no\n";

last;

}

}

close($fh);
```

22.Database interface using perl dbi.

DBI – DataBase Interface module (more precisely a collection of modules) is a feature rich, efficient and yet simple tool to access databases using perl. Almost all Linux distributions have them, if not you can download from cpan.org. The interface to any DBMS requires two sets of tools – one DBI itself which is generic, two the DBD::the_database. DBD is the driver component and you should install drivers for whatever database you are using. DBI drivers are available for almost all standard databases.

Normally database access workflow is like this:

a.Connect to the database (logging) using username,password etc.. Once properly authenticated a database-handle will be given.

b.Create the sql query, use database-handle to send the query to the server and ask it to prepare the query.

c. Server parses the sql, if no errors, returns a statement-handle.

d.Use the statement-handle to execute the query.

f.Use statement-handle to fetch data – single row or multiple rows at a time.

g.Close the statement handle

h. Repeat steps b to g as long as you want, with new queries

i.Finally disconnect from database(logout) using database-handle.

Let us see them by means of a sample code.

Assumptions:

DBD driver for mysql is already installed.

Database Server: mysql running on local host

Database user name: test Password: test123

Database name : testdb

Table : names

Columns in the table: id,name,age

Example 1:

1. List all records in the table and find out average age

Let us start the code. (codes are give in italics and bold)

Step 1:

Connecting to the database

use DBI;

my $dbh=DBI->connect(‘DBI:mysql:database=testdb;host=localhost’,'test’,'test123′);

Connect requires three arguments : datasource, username,password

First argument -datasource gives information about the database server like type of dbms, location etc.

In our example datasource is specified as DBI:mysql:database=testdb;host=localhost

Here DBI:mysql means use mysql driver.

database=testdb means use the database testdb.

host=localhost means the host in which the database is running.

Other two arguments are username and password which need no explanation.

Step 2

Run the select query on the server.

First store sql in a variable like this

my $query='select * from name ';

Then send the sql to the server for parsing and checking

my $sth=$dbh->prepare($query) or die "could not prepare $query\n";

In the above statement

$dbh is the database connection handle we got while using DBI->connect earlier.

$sth is the statement handle returned upon successful preparation.

$query refers to the sql statement. The query can be given directly as string also.

Here we do some error checking by using die. The $sth that is returned will be required for any further opertion on this query.

Now we will run the query on the server

$sth->execute();

Note here, we are simply using $sth to run the query. Once we call execute, the server runs the query and keeps the result set ready for retrieval.

Step 3

Get results from the server one row at a time.

fetchrow_array() is a function that will return one row of data and store result in an array.

We will use a while loop to fetch all rows from the server.

while (($id,$name,$age)=$sth->fetchrow_array()){

print "id=$id name=$name age=$age\n";

}

$sth->fetchrow will return a null when there are no more rows. Thus this loop will run until null.

($id,$name,$age)=$sth->fetchrow_array() is used to equate the rows returned to a set of variables.

Step 4

Close the statement handle

$sth->finish();

Step 5

Close the database connection

$dbh->disconnect();

Here is the output of running the script.

id=1 name=RAMAN age=45

id=2 name=RAVI age=35

For the sake convenience I am repeating program listing here.

use DBI;

my $dbh=DBI->connect('DBI:mysql:database=testdb;host=localhost','test','test123′);

my $query='select * from name ';

my $sth=$dbh->prepare($query) or die "could not prepare $query\n";

```
$sth->execute();

while (($id,$name,$age)=$sth->fetchrow_array()){

print “id=$id name=$name age=$age\n”;

}

$sth->finish();

$dbh->disconnect()
```

Example 2.

Insert a record accepting input from terminal

Here is the code to insert a row into name table.

Step 1:

Establishing connection with database. For explanation see previous example

```
use DBI;

my $dbh=DBI->connect(‘DBI:mysql:database=testdb;host=localhost’,'test’,'test123′);
```

Step 2

Accept input from keyboard

```
print ‘Enter id:’;

$id=<STDIN>;

print ‘Enter Name:’;

$name=<STDIN>;

print ‘Enter age’;

$age=<STDIN>;
```

use chomp to remove any newlines

```
chomp $id;
```

chomp $age;

chomp $name;

Explanation : The print statement is simple – just shows message on the screen.

$id=<STDIN> means accept value from standard input (by default keyboard) and store the value in the variable $id;

Step 3

Create the Sql statement

$query=sprintf "insert into name(id,name,age) values(%d,%s,%d)",$id,$dbh->quote($name),$age;

Explanation: Here we create a string using sprintf function, by variable substitution.

The sprintf in perl is similar to sprintf in C.

$dbh->quote() function is used on string values, so that quotes are properly taken care of -e.g names like D'Silva. It is a best practice to use this.

At the end of this line the $query variable will have an sql which is an insert statement.

Step 4.

Run the query on the server and insert data.

$dbh->do($query) or die "could not do $query\n";

The sql execution in the previous example was done in three stages namely prepare, execute,fetchrow_array. In the case of insert statement no rows are returned. Hence, we can use do(), which combines all three stages into one. It will return number of rows affected.

Step 5.

Disconnect from the server

$dbh->disconnect();

For the sake of convenience the code is repeated here.

use DBI;

```
my $dbh=DBI-
>connect('DBI:mysql:database=testdb;host=localhost','test','test123′);

print 'Enter id:';

$id=<STDIN>;

print 'Enter Name:';

$name=<STDIN>;

print 'Enter age';

$age=<STDIN>;

chomp $id;

chomp $age;

chomp $name;

$query=sprintf "insert into name(id,name,age) values(%d,%s,%d)",$id,$dbh-
>quote($name),$age;

$dbh->do($query) or die "could not do $query\n";

$dbh->disconnect();
```

As you can see from the above examples dbi is pretty simple. Same code can be used for any database. Only change required will be in connection step.

The performance of dbi is very good, and it has lot features like fetching column names, types etc.

Exercise

1.Accept a name from the user. Then list all records matching that name.

2. Accept id and delete the record matching this id.