

NAME: - LIZA KANJI PATEL

CLASS: - M.Sc. PART - II

COURSE: - BIOINFORMATICS

ACADEMIC YEAR: - 2022-2023

ROLL NO.: - 117

PAPER CODE: - GNKPSB|P304

COURSE TITLE: - INTRODUCTION TO PERL AND MONGODB

GURU NANAK KHALSA COLLEGE

MATUNGA, MUMBAI-400 019.

DEPARTMENT OF BIOINFORMATICS

CERTIFICATE

This is to certify that **Ms. Liza Kanji Patel (Roll.No.113)** of M.Sc. Part II Bioinformatics has satisfactorily completed the practical Semester III course prescribed by the University of Mumbai during the academic year 2022-2023.

TEACHER INCHARGE

HEAD OF DEPARTMENT

INDEX:

SR.NO.	EXPERIMENTS	PAGE NO.	DATE	SIGN
1.	Simple programs on Variable types used in Perl	1-3	09/07/22	
2.	Conditional statements and Loops	4-19	09/07/22	
3.	Operators used on scalar, array and hash variables	20-30	20/07/22	
4.	Subroutine	31-34	01/08/22	
5.	References and Dereferences, and Scope of variables	35-42	01/08/22	
6.	Regular Expression	43-48	03/08/22	
7.	Metacharacters, Quantifiers and Substring	49-63	18/08/22	
8.	Perl Formatting	64-76	12/09/22	
9.	OOPs in perl	77-86	09/10/22	
10.	DBI in perl	87-97	08/10/22	
11.	File handling and directory	98-108	10/10/22	
12.	MongoDB basic commands	109-124	19/11/22	

Practical No. 1

Simple programs on Variable types used in Perl

AIM:

To understand and write Simple programs on Variable types used in Perl

THEORY:

Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more.

What is Perl?

- Perl is a stable, cross platform programming language.
- Though Perl is not officially an acronym but few people used it as **Practical Extraction and Report Language**.
- It is used for mission critical projects in the public and private sectors.
- Perl is an *Open Source* software, licensed under its *Artistic License*, or the *GNU General Public License (GPL)*.
- Perl was created by Larry Wall.
-

Perl has three basic data types: scalars, arrays of scalars, and hashes of scalars, also known as associative arrays

Scalar

Scalars are simple variables. They are preceded by a dollar sign (\$). A scalar is either a number, a string, or a reference. A reference is actually an address of a variable.

```
$age = 25;
```

Arrays

Arrays are ordered lists of scalars that you access with a numeric index, which starts with 0. They are preceded by an "at" sign (@).

```
@ages = (25, 30, 40);
```

```
@names = ("John Paul", "Lisa", "Kumar");
```

Hashes

Hashes are unordered sets of key/value pairs that you access using the keys as subscripts. They are preceded by a percent sign (%).

```
%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);
```

Q.1) Write a Perl script to store DNA sequence in scalar variable entered by user and display an output.

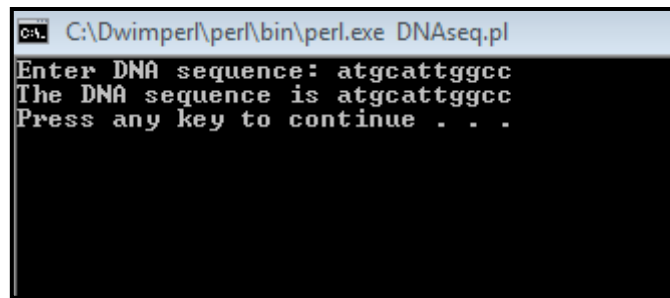
CODE:

```
print "Enter DNA sequence: ";
```

```
$seq = <stdin>;
```

```
print "The DNA sequence is $seq";
```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimper\perl\bin\perl.exe DNAseq.pl'. The prompt is 'C:\>'. The user has entered 'atgcattggcc' in response to the prompt 'Enter DNA sequence:'. The script has printed 'The DNA sequence is atgcattggcc' and 'Press any key to continue . . .'.

```
C:\> C:\Dwimper\perl\bin\perl.exe DNAseq.pl
Enter DNA sequence: atgcattggcc
The DNA sequence is atgcattggcc
Press any key to continue . . .
```

Fig1. Output for Perl script to store DNA sequence in scalar variable entered by user

Q.2) Write a Perl script to ask user to enter RNA sequence using an array without loops (5 sequence)

CODE:

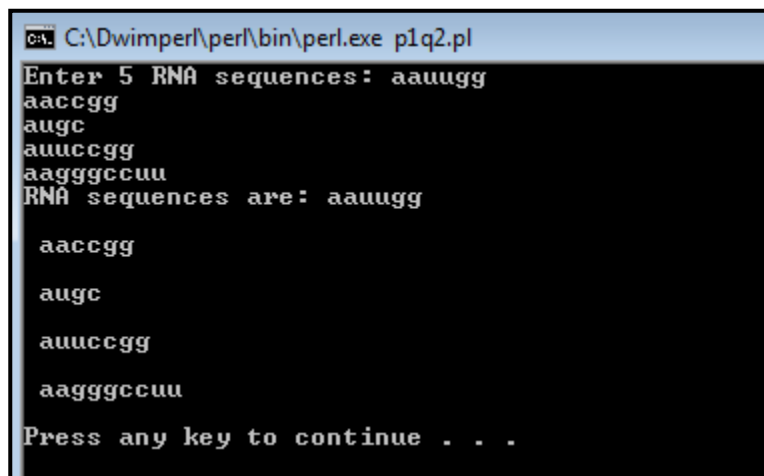
```
@sequence;

print "Enter 5 RNA sequences: ";

$sequence[0] = <stdin>;
$sequence[1] = <stdin>;
$sequence[2] = <stdin>;
$sequence[3] = <stdin>;
$sequence[4] = <stdin>;

print "RNA sequences are: $sequence[0]\n $sequence[1]\n $sequence[2]\n $sequence[3]\n $sequence[4]\n";
```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimper\perl\bin\perl.exe p1q2.pl'. The prompt is 'C:\>'. The user has entered five RNA sequences: 'aauggg', 'aaccgg', 'augc', 'auuccgg', and 'aagggccuu'. The script has printed 'Enter 5 RNA sequences:' followed by the five sequences on separate lines. Then it has printed 'RNA sequences are:' followed by the same five sequences on separate lines. Finally, it has printed 'Press any key to continue . . .'.

```
C:\> C:\Dwimper\perl\bin\perl.exe p1q2.pl
Enter 5 RNA sequences: aauggg
aaccgg
augc
auuccgg
aagggccuu
RNA sequences are: aauggg
aaccgg
augc
auuccgg
aagggccuu
Press any key to continue . . .
```

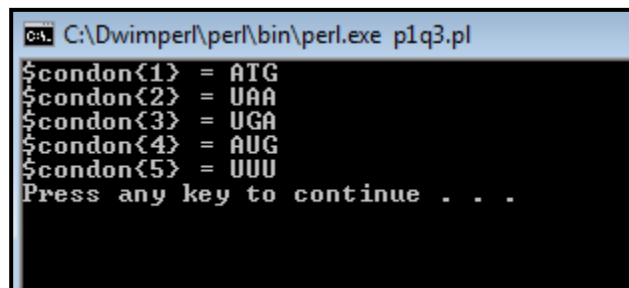
Fig2. Output for Perl script to ask user to enter RNA sequence using an array without loops

Q.3) Write a Perl script to store codon using hash variables.

CODE:

```
%codon = ('1' => ATG, '2' => UAA, '3' => UGA, '4' => AUG, '5' => UUU);  
  
print "\$condon{1} = $codon{1}\n";  
  
print "\$condon{2} = $codon{2}\n";  
  
print "\$condon{3} = $codon{3}\n";  
  
print "\$condon{4} = $codon{4}\n";  
  
print "\$condon{5} = $codon{5}\n";
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe plq3.pl  
$condon{1} = ATG  
$condon{2} = UAA  
$condon{3} = UGA  
$condon{4} = AUG  
$condon{5} = UUU  
Press any key to continue . . .
```

Fig3. Output for Perl script to store codon using hash variables.

Practical No. 2

Conditional statements and Loops

AIM:

To understand and write perl programs for Conditional statements and Loops

THEROY:

Perl conditional statements helps in the decision making, which require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Perl programming language provides the following types of conditional statements.

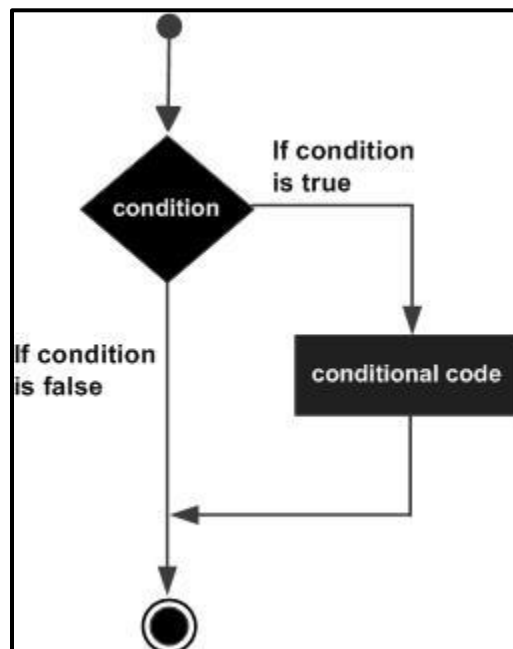
A Perl **if** statement consists of a boolean expression followed by one or more statements.

Syntax

The syntax of an **if** statement in Perl programming language is –

```
if(boolean_expression) {  
# statement(s) will execute if the given condition is true  
}
```

If the boolean expression evaluates to **true** then the block of code inside the **if** statement will be executed. If boolean expression evaluates to **false** then the first set of code after the end of the **if** statement (after the closing curly brace) will be executed.



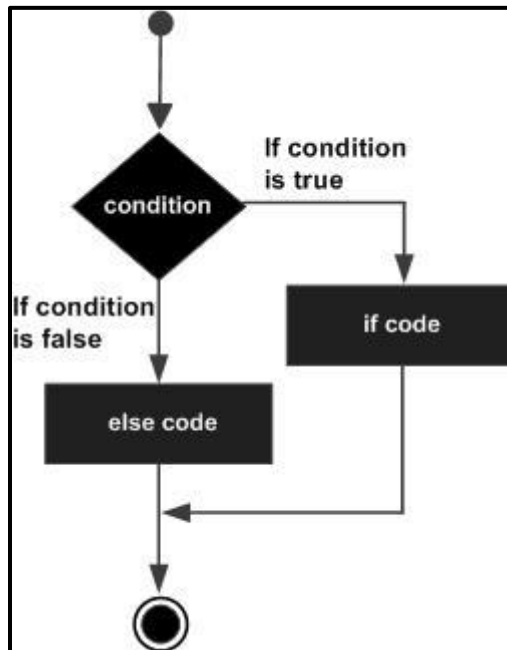
A Perl **if** statement can be followed by an optional **else** statement, which executes when the boolean expression is false.

Syntax

The syntax of an **if...else** statement in Perl programming language is –

```
if(boolean_expression) {  
# statement(s) will execute if the given condition is true  
} else {  
# statement(s) will execute if the given condition is false  
}
```

If the boolean expression evaluates to **true**, then the **if block** of code will be executed otherwise **else block** of code will be executed.



An **if** statement can be followed by an optional **elsif...else** statement, which is very useful to test the various conditions using single if...elsif statement.

When using **if**, **elsif**, **else** statements there are few points to keep in mind.

- An **if** can have zero or one **else**'s and it must come after any **elsif**'s.
- An **if** can have zero to many **elsif**'s and they must come before the **else**.
- Once an **elsif** succeeds, none of the remaining **elsif**'s or **else**'s will be tested.

Syntax

The syntax of an **if...elsif...else** statement in Perl programming language is –

```
if(boolean_expression 1) {  
# Executes when the boolean expression 1 is true  
} elsif( boolean_expression 2) {  
# Executes when the boolean expression 2 is true  
} elsif( boolean_expression 3) {  
# Executes when the boolean expression 3 is true  
} else {
```



```
# Executes when the none of the above condition is true
}
```

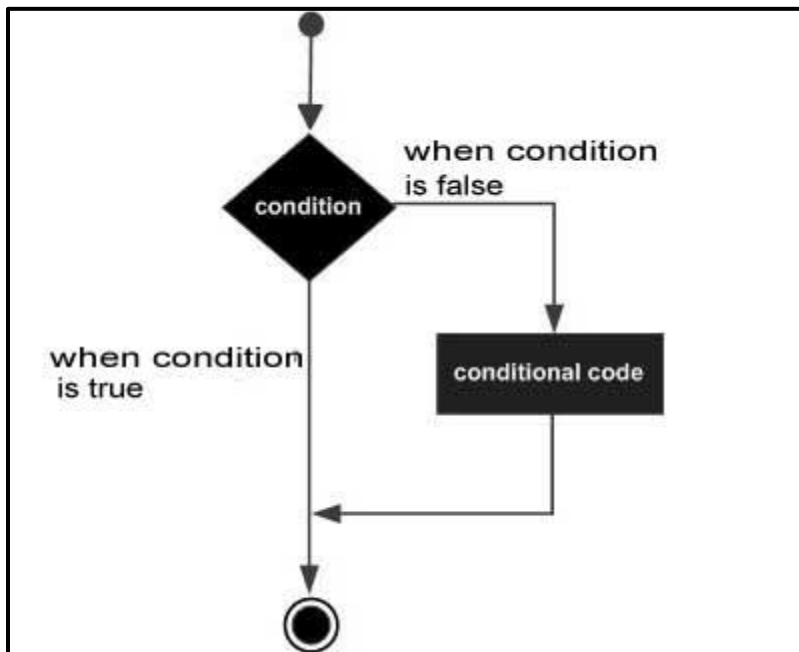
A Perl **unless** statement consists of a boolean expression followed by one or more statements.

Syntax

The syntax of an unless statement in Perl programming language is –

```
unless(boolean_expression) {
# statement(s) will execute if the given condition is false
}
```

If the boolean expression evaluates to **false**, then the block of code inside the unless statement will be executed. If boolean expression evaluates to **true** then the first set of code after the end of the unless statement (after the closing curly brace) will be executed.



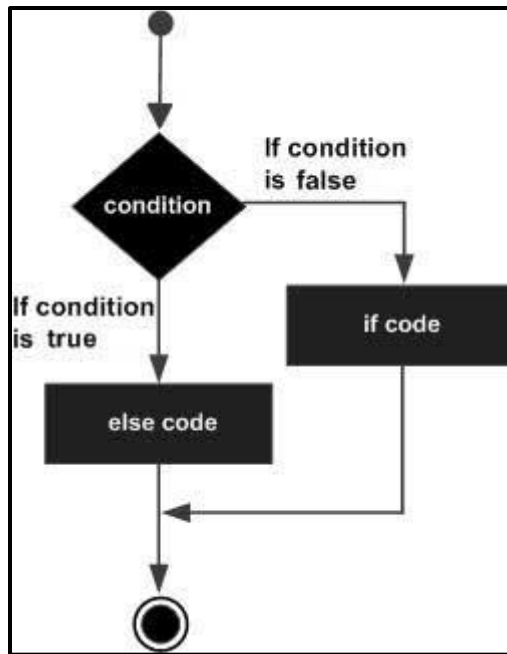
A Perl **unless** statement can be followed by an optional **else** statement, which executes when the boolean expression is true.

Syntax

The syntax of an **unless...else** statement in Perl programming language is –

```
unless(boolean_expression) {
# statement(s) will execute if the given condition is false
} else {
# statement(s) will execute if the given condition is true
}
```

If the boolean expression evaluates to **true** then the **unless block** of code will be executed otherwise **else block** of code will be executed.



An **unless** statement can be followed by an optional **elsif...else** statement, which is very useful to test the various conditions using single **unless...elsif** statement.

When using **unless**, **elsif**, **else** statements there are few points to keep in mind.

- An **unless** can have zero or one **else**'s and it must come after any **elsif**'s.
- An **unless** can have zero to many **elsif**'s and they must come before the **else**.
- Once an **elsif** succeeds, none of the remaining **elsif**'s or **else**'s will be tested.

Syntax

The syntax of an **unless...elsif...else** statement in Perl programming language is –

```

unless(boolean_expression 1) {
# Executes when the boolean expression 1 is false
} elsif( boolean_expression 2) {
# Executes when the boolean expression 2 is true
} elsif( boolean_expression 3) {
# Executes when the boolean expression 3 is true
} else {
# Executes when the none of the above condition is met
}
  
```

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

A switch case implementation is dependent on **Switch** module and **Switch** module has been implemented using *Filter::Util::Call* and *Text::Balanced* and requires both these modules to be installed.

Syntax

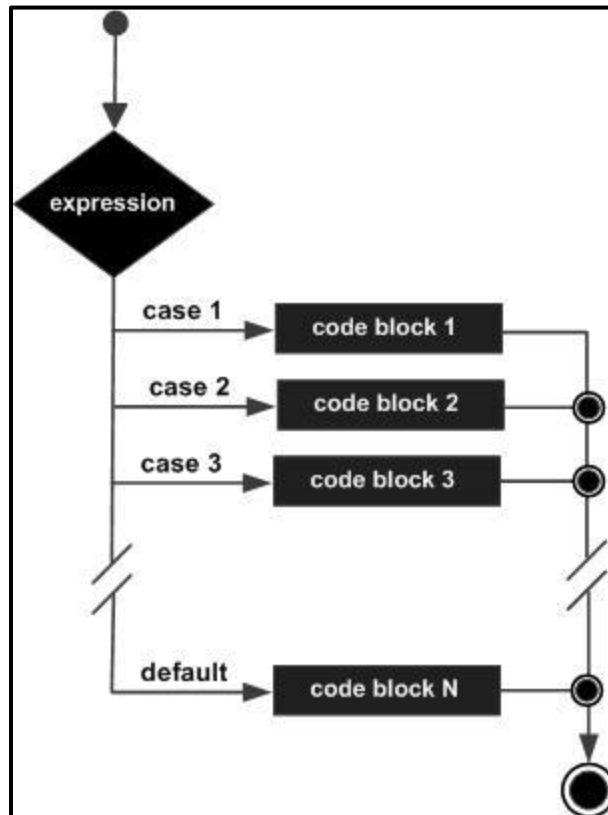
The synopsis for a **switch** statement in Perl programming language is as follows –

use Switch;

```

switch(argument) {
case 1 { print "number 1" }
case "a" { print "string a" }
case [1..10,42] { print "number in list" }
case (\@array) { print "number in list" }
case /\w+/ { print "pattern" }
case qr/\w+/ { print "pattern" }
case (\%hash) { print "entry in hash" }
case (\&sub) { print "arg to subroutine" }
else { print "previous case not true" }
}

```



Loops:

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times

Perl programming language provides the following types of loop to handle the looping requirements.

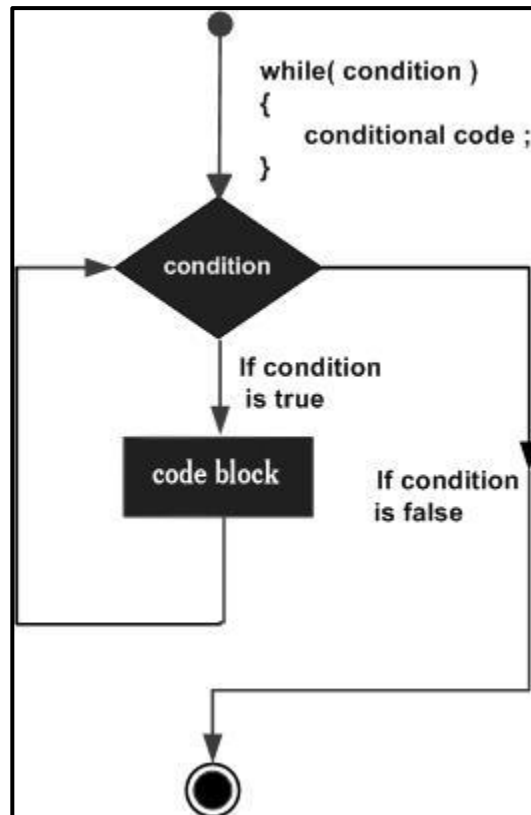
A **while** loop statement in Perl programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a **while** loop in Perl programming language is –

```
while(condition) {  
statement(s);  
}
```

Here **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.



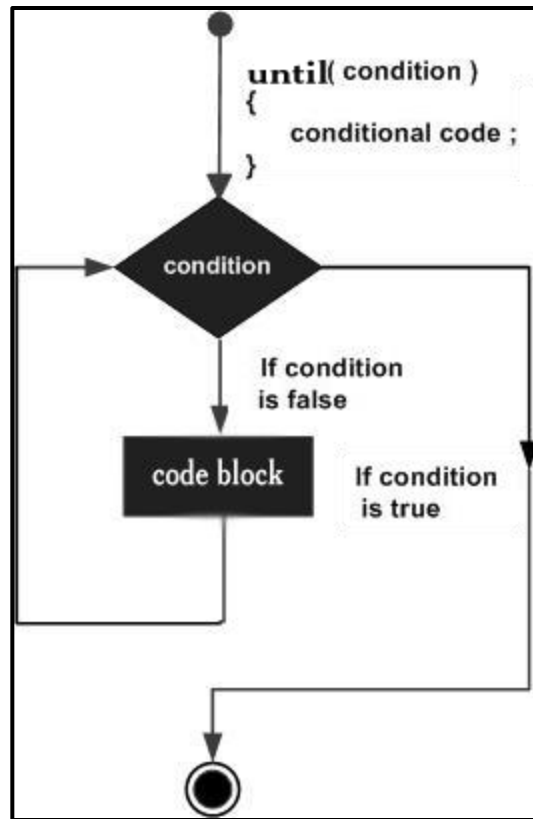
An **until** loop statement in Perl programming language repeatedly executes a target statement as long as a given condition is false.

Syntax

The syntax of an **until** loop in Perl programming language is –

```
until(condition) {  
statement(s);  
}
```

Here **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression. The loop iterates until the condition becomes true. When the condition becomes true, the program control passes to the line immediately following the loop.



A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

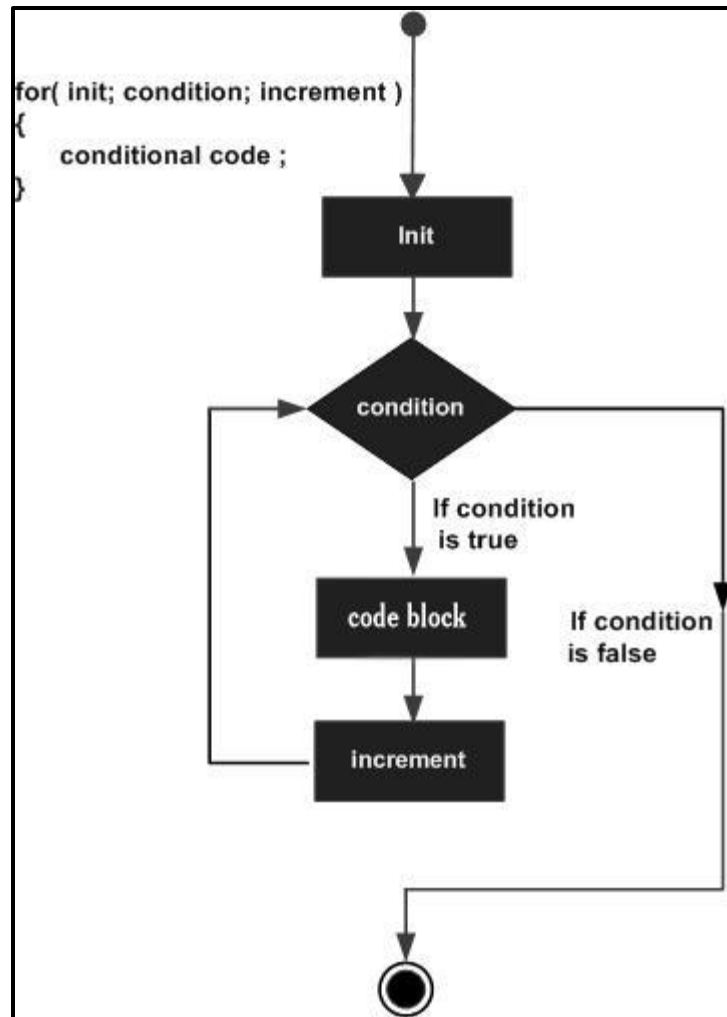
The syntax of a **for** loop in Perl programming language is –

```

for ( init; condition; increment ) {
statement(s);
}
  
```

Here is the flow of control in a **for** loop –

- The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- After the body of the for loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop terminates.

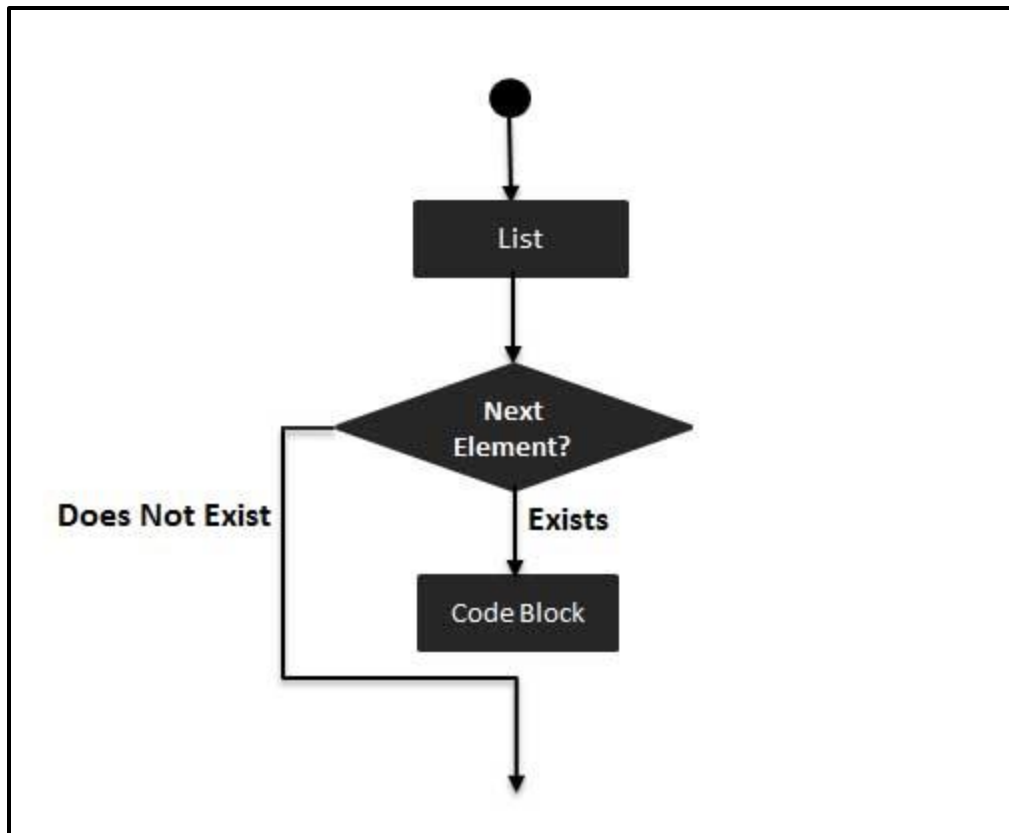


The **foreach** loop iterates over a list value and sets the control variable (var) to be each element of the list in turn –

Syntax

The syntax of a **foreach** loop in Perl programming language is –

```
foreach var (list) {  
  ...  
}
```



Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop checks its condition at the bottom of the loop.

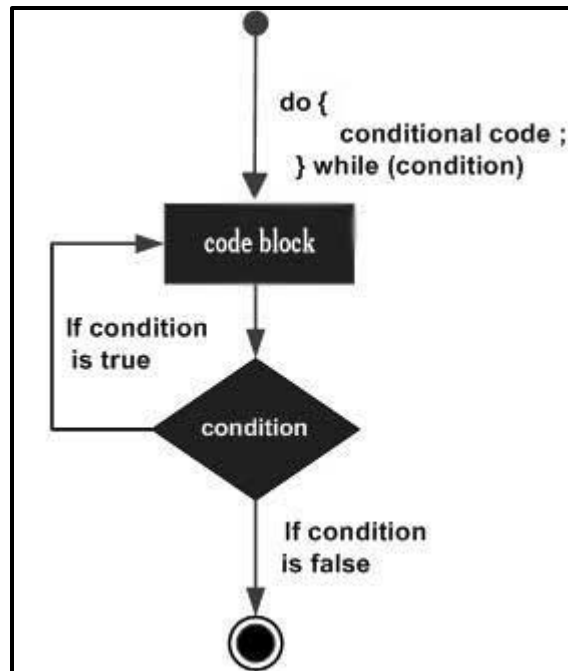
A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax

The syntax of a **do...while** loop in Perl is –

```
do {  
statement(s);  
}while( condition );
```

It should be noted that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested. If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.



A loop can be nested inside of another loop. Perl allows to nest all type of loops to be nested.

Syntax

The syntax for a **nested for loop** statement in Perl is as follows –

```

for ( init; condition; increment ) {
  for ( init; condition; increment ) {
    statement(s);
  }
  statement(s);
}

```

Q.1) Write a Perl script to ask user to enter a number and check whether entering number is even or odd.

CODE:

```

print "Enter a number: ";

$num = <stdin>;

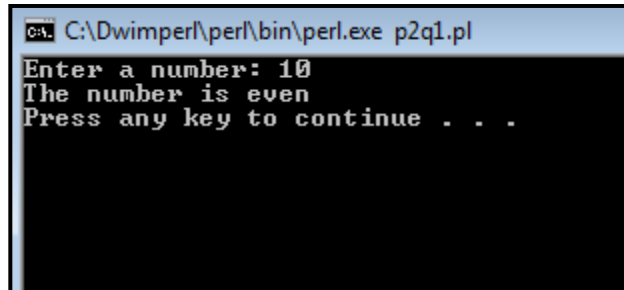
if ($num%2 == 0)
{
    print "The number is even\n";
}
else
{

```



```
print "The number is odd\n";  
}
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p2q1.pl  
Enter a number: 10  
The number is even  
Press any key to continue . . .
```

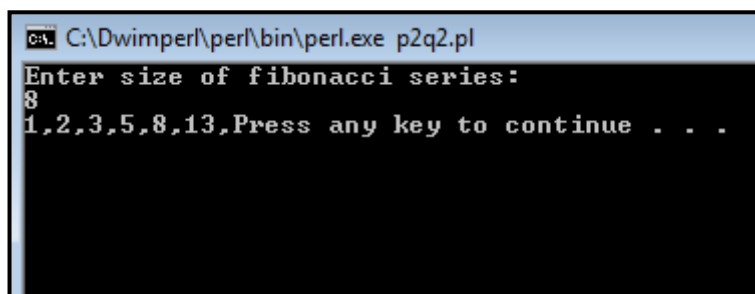
Fig1. Output for Perl script to ask user to enter a number and check whether entering number is even or odd.

Q.2) Write a Perl script to ask user to enter number to display Fibonacci series.

CODE:

```
print "Enter size of fibonacci series: \n";  
$size = <stdin>;  
$f=0;  
$s=1;  
for($i=1;$i<=($size-2);$i++)  
{  
    $t = $f+$s;  
    print "$t,";  
    $f = $s;  
    $s = $t;  
}
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p2q2.pl  
Enter size of fibonacci series:  
8  
1,2,3,5,8,13,Press any key to continue . . .
```

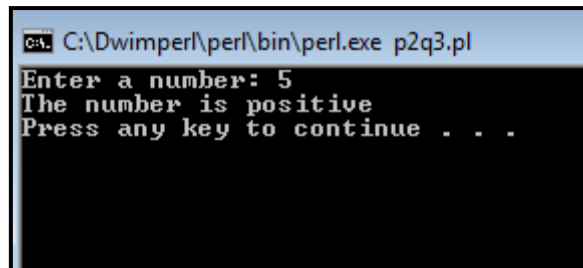
Fig2. Output for Perl script to ask user to enter number to display Fibonacci series.

Q.3) Write a Perl script to ask user to enter a number and check whether entering number is negative or positive.

CODE:

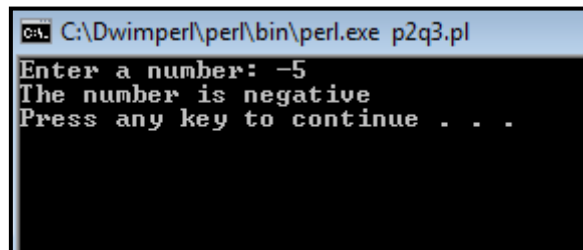
```
print "Enter a number: ";  
  
$num = <stdin>;  
  
if ($num < 0)  
{  
    print "The number is negative\n";  
}  
  
else  
{  
    print "The number is positive\n";  
}
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p2q3.pl  
Enter a number: 5  
The number is positive  
Press any key to continue . . .
```

Fig3.1. Output for Perl script to ask user to enter a number and check whether entering number is negative or positive.



```
C:\Dwimperl\perl\bin\perl.exe p2q3.pl  
Enter a number: -5  
The number is negative  
Press any key to continue . . .
```

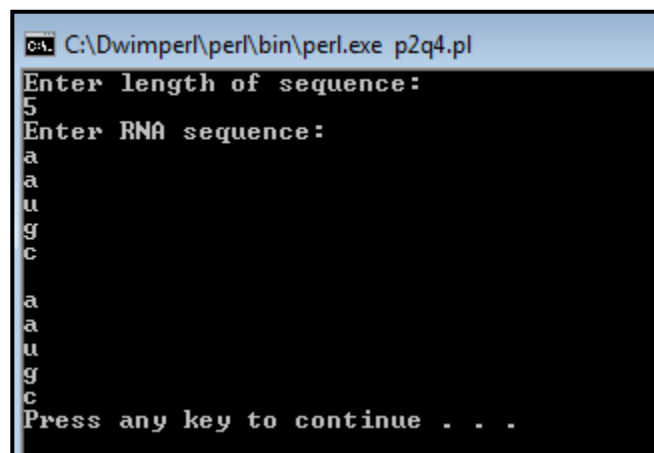
Fig3.2. Output for Perl script to ask user to enter a number and check whether entering number is negative or positive.

Q.4) Write a Perl script to ask user to enter RNA sequence using an array with for and foreach loops.

CODE:

```
@seq;  
print "Enter length of sequence: \n";  
$length = <stdin>;  
print "Enter RNA sequence: \n";  
for ($i=0; $i<$length; $i++)  
{  
    $seq[$i] = <stdin>;  
}  
print "\n";  
foreach $n(@seq)  
{  
    print $n;  
}
```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimperl\perl\bin\perl.exe p2q4.pl'. The prompt displays the following text: 'Enter length of sequence:', followed by the input '5'. Then it shows 'Enter RNA sequence:', followed by the input 'a', 'a', 'u', 'g', 'c' on separate lines. At the bottom, it says 'Press any key to continue . . .'.

```
C:\Dwimperl\perl\bin\perl.exe p2q4.pl  
Enter length of sequence:  
5  
Enter RNA sequence:  
a  
a  
u  
g  
c  
Press any key to continue . . .
```

Fig4. Output for a Perl script to ask user to enter RNA sequence using an array with for and foreach loops.

Q.5) Write a Perl script to ask user to enter DNA sequence using a hashes and display keys and values separately.

CODE:

```
%seq;

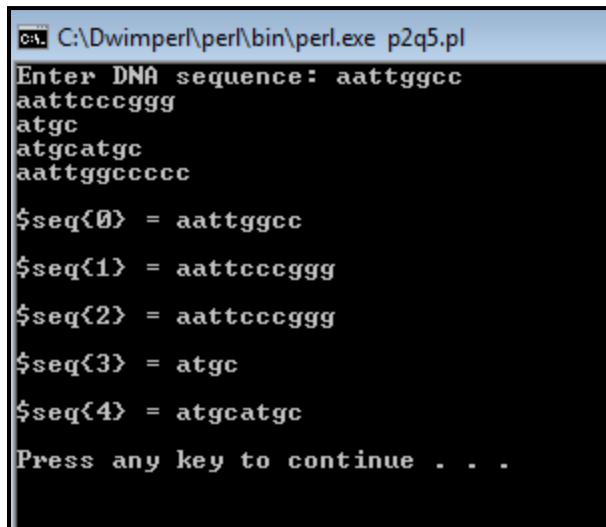
print "Enter DNA sequence: ";

for ($i=0; $i<5; $i++)
{
    $seq{$i} = <stdin>;
}

print "\n";

print "\$seq{0} = $seq{0}\n";
print "\$seq{1} = $seq{1}\n";
print "\$seq{2} = $seq{1}\n";
print "\$seq{3} = $seq{2}\n";
print "\$seq{4} = $seq{3}\n";
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p2q5.pl
Enter DNA sequence: aattggcc
aattcccggg
atgc
atgcatgc
aattggccccc

$seq{0} = aattggcc
$seq{1} = aattcccggg
$seq{2} = aattcccggg
$seq{3} = atgc
$seq{4} = atgcatgc
Press any key to continue . . .
```

Fig5. Output for a Perl script to ask user to enter DNA sequence using a hashes and display keys and values separately.

Q.6) Write a Perl script to ask user to enter number and find factorial of an entered number.

CODE:

```
print "Enter a number: ";  
$num = <stdin>;  
$factorial = 1;  
for ($i=1; $i<=$num; $i++)  
{  
    $factorial = $factorial*$i;  
}  
print "Factorial of the given number is $factorial\n";
```

OUTPUT:

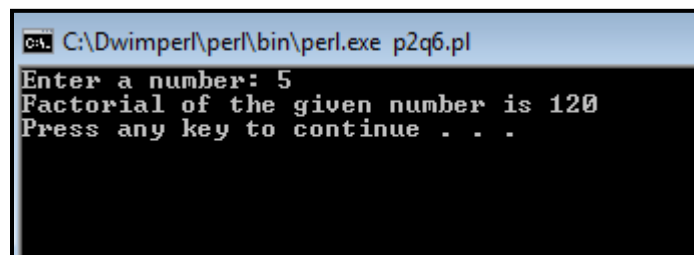


Fig6. Output for a Perl script to ask user to enter number and find factorial of an entered number.

Q.7) Write a Perl script to store DNA sequence (by user) and check entered sequence is DNA or not.

```
print "Enter a sequence: ";  
$dna = <stdin>;  
  
if($dna eq "a\n"){  
    print "The given sequence is DNA";  
}  
elsif($dna eq "t\n"){  
    print "The given sequence is DNA";  
}  
elsif($dna eq "g\n"){  
    print "The given sequence is DNA";  
}  
elsif($dna eq "c\n"){  
    print "The given sequence is DNA";  
}
```

```

}else{

    print "The given sequence is not DNA";

}

```

OUTPUT:

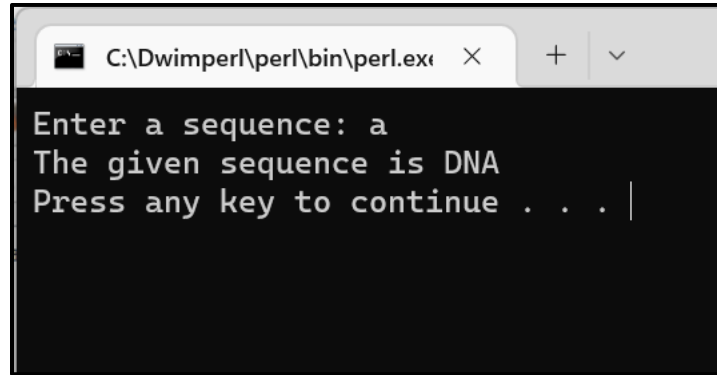


Fig7. Output for Perl script to store DNA sequence (by user) and check entered sequence is DNA or not

Q.8) Write a Perl script to display following pattern:

A A A A A

A A A A

A A A

A A

A

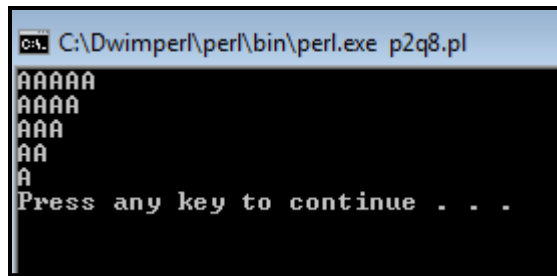
CODE:

```

for($i=4; $i>=0; $i--)
{
    for($j=0; $j<=$i; $j++)
    {
        print 'A';
    }
    print "\n"
}

```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p2q8.pl
AAAAA
AAAA
AAA
AA
A
Press any key to continue . . .
```

Fig8. Output for Perl script to display a pattern.

Practical No 3

Operators used on scalar, array and hash variables

AIM:

To understand and write perl programs for operators used on scalar, array and hash variables

THEORY:

What is an Operator?

Simple answer can be given using the expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. Perl language supports many operator types, but following is a list of important and most frequently used operators –

- Arithmetic Operators
- Logical Operators
- Equality Operators
- Miscellaneous Operators

Perl Arithmetic Operators

Assume variable \$a holds 10 and variable \$b holds 20, then following are the Perl arithmetic operators –

+ (Addition)

Adds values on either side of the operator

Example – \$a + \$b will give 30

- (Subtraction)

Subtracts right hand operand from left hand operand

Example – \$a - \$b will give -10

* (Multiplication)

Multiplies values on either side of the operator

Example – \$a * \$b will give 200

/ (Division)

Divides left hand operand by right hand operand

Example – \$b / \$a will give 2

% (Modulus)

Divides left hand operand by right hand operand and returns remainder

Example – \$b % \$a will give 0

**** (Exponent)**

Performs exponential (power) calculation on operators

Example – \$a**\$b will give 10 to the power 20

Perl Equality Operators

These are also called relational operators. Assume variable \$a holds 10 and variable \$b holds 20 then, lets check the following numeric equality operators –

== (equal to)

Checks if the value of two operands are equal or not, if yes then condition becomes true.

Example – (\$a == \$b) is not true.

== (equal to)

Checks if the value of two operands are equal or not, if yes then condition becomes true.

Example – (\$a == \$b) is not true.

<=>

Checks if the value of two operands are equal or not, and returns -1, 0, or 1 depending on whether the left argument is numerically less than, equal to, or greater than the right argument.

Example – (\$a <=> \$b) returns -1.

> (greater than)

Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.

Example – (\$a > \$b) is not true.

< (less than)

Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.

Example – (\$a < \$b) is true

< (less than)

Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.

Example – (\$a < \$b) is true

<= (less than or equal to)

Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

Example – (\$a <= \$b) is true.

Below is a list of equity operators. Assume variable \$a holds "abc" and variable \$b holds "xyz" then, lets check the following string equality operators –

lt

Returns true if the left argument is stringwise less than the right argument.

Example – (\$a lt \$b) is true.

gt

Returns true if the left argument is stringwise greater than the right argument.

Example – (\$a gt \$b) is false.

le

Returns true if the left argument is stringwise less than or equal to the right argument.

Example – (\$a le \$b) is true.

ge

Returns true if the left argument is stringwise greater than or equal to the right argument.

Example – (\$a ge \$b) is false.

eq

Returns true if the left argument is stringwise equal to the right argument.

Example – (\$a eq \$b) is false.

ne

Returns true if the left argument is stringwise not equal to the right argument.

Example – (\$a ne \$b) is true.

cmp

Returns -1, 0, or 1 depending on whether the left argument is stringwise less than, equal to, or greater than the right argument.

Example – (\$a cmp \$b) is -1.

Perl Logical Operators

There are following logical operators supported by Perl language. Assume variable \$a holds true and variable \$b holds false then –

and

Called Logical AND operator. If both the operands are true then then condition becomes true.

Example – (\$a and \$b) is false.

&&

C-style Logical AND operator copies a bit to the result if it exists in both operands.

Example – (\$a && \$b) is false.

or

Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.

Example – (\$a or \$b) is true.

||

C-style Logical OR operator copies a bit if it exists in either operand.

Example – (\$a || \$b) is true.

not

Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.

Example – not(\$a and \$b) is true.

Miscellaneous Operators

There are following miscellaneous operators supported by Perl language. Assume variable a holds 10 and variable b holds 20 then –

.

Binary operator dot (.) concatenates two strings.

Example – If \$a = "abc", \$b = "def" then \$a.\$b will give "abcdef"

x

The repetition operator x returns a string consisting of the left operand repeated the number of times specified by the right operand.

Example – ('-' x 3) will give ---.

x

The repetition operator x returns a string consisting of the left operand repeated the number of times specified by the right operand.

Example – ('-' x 3) will give ---.

++

Auto Increment operator increases integer value by one

Example – \$a++ will give 11

--

Auto Decrement operator decreases integer value by one

Example – \$a-- will give 9

->

The arrow operator is mostly used in dereferencing a method or variable from an object or a class name

Example – \$obj->\$a is an example to access variable \$a from object \$obj.

Q.1) Write a Perl script accept two number and a string and perform the following operations:

a. Perl Arithmetic Operators

b. Miscellaneous Operators

CODE:

```
print "Enter two numbers:\n";
```

```
$num1 = <stdin>;
```

```

$num2 = <stdin>;

$add = $num1 + $num2;

print "Addition of numbers = $add\n";

$sub = $num1 - $num2;

print "Subtraction of numbers = $sub\n";

$mul = $num1 * $num2;

print "Multiplication of numbers = $mul\n";

$div = $num1 / $num2;

print "Divide of numbers = $div\n";

$mod = $num1 % $num2;

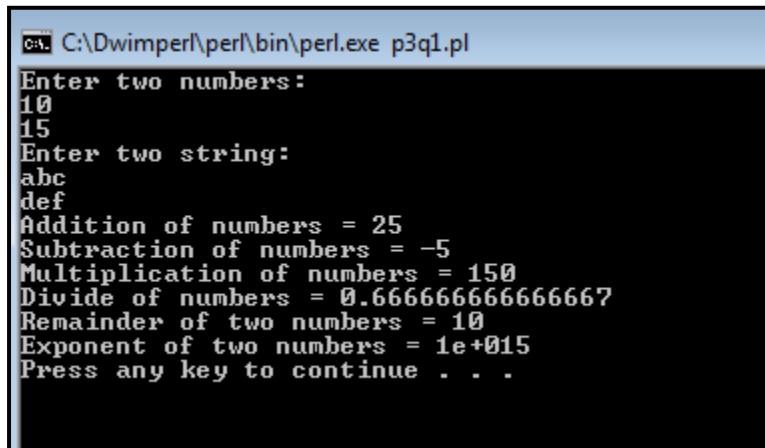
print "Remainder of two numbers = $mod\n";

$exp = $num1**$num2;

print "Exponent of two numbers = $exp\n";

```

OUTPUT:



```

C:\Dwimper\perl\bin\perl.exe p3q1.pl
Enter two numbers:
10
15
Enter two string:
abc
def
Addition of numbers = 25
Subtraction of numbers = -5
Multiplication of numbers = 150
Divide of numbers = 0.6666666666666667
Remainder of two numbers = 10
Exponent of two numbers = 1e+015
Press any key to continue . . .

```

Fig1a. Output for perl Arithmetic Operators.

```

print "Enter two string:\n";

$string1 = <stdin>;

$string2 = <stdin>;

print "Repetative operator\n";

print ($string1 x 3);

print "Dot operator\n";

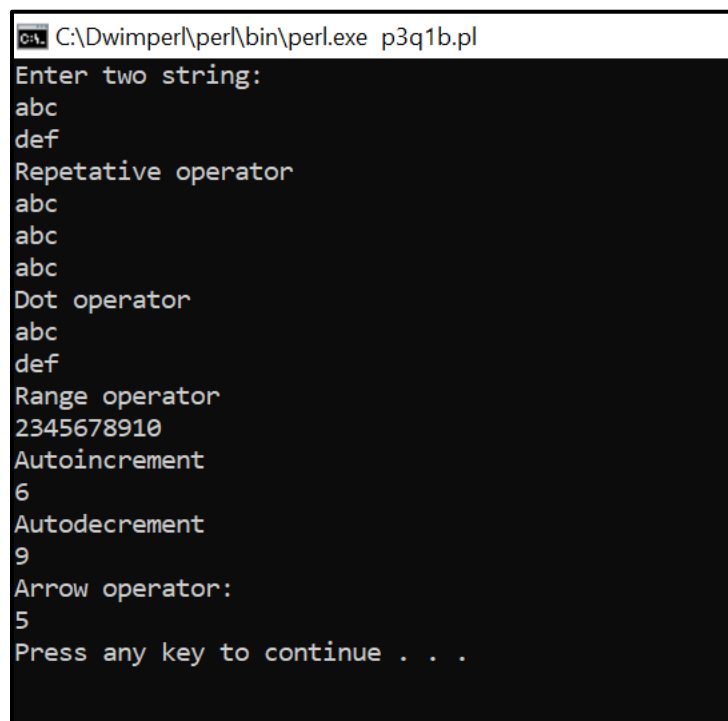
```

```

print $string1.$string2;
print "Range operator\n";
print (2..10);
print "\n";
print "Autoincrement\n";
$a = 5;
$b = ++$a;
print "$b\n";
print "Autodecrement\n";
$b=10;
$c = --$b;
print "$c\n";
print "Arrow operator:\n";
$arr = [2, 3, 5, 7, 11];
print "$arr->[2]\n";

```

OUTPUT:



```

C:\Dwimper\perl\bin\perl.exe p3q1b.pl
Enter two string:
abc
def
Repetative operator
abc
abc
abc
Dot operator
abc
def
Range operator
2345678910
Autoincrement
6
Autodecrement
9
Arrow operator:
5
Press any key to continue . . .

```

Fig1b. Output for perl miscellaneous operators

Q.2) Write a Perl script to accept three number and display smallest number.

CODE:

```
print "Enter 3 numbers: \n";

$a = <stdin>;

$b = <stdin>;

$c = <stdin>;

if ($a <= $b && $a <= $c)

{

    print "$a is the smallest number\n";

}elsif ($b <= $a && $b <= $c)

{

    print "$b is the smallest number\n";

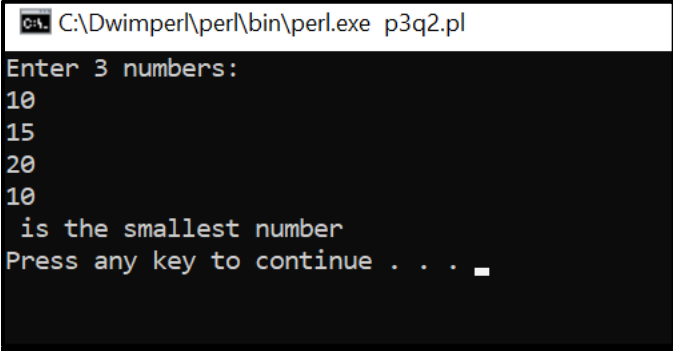
}else

{

    print "$c is the smallest number\n";

}
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p3q2.pl
Enter 3 numbers:
10
15
20
10
 is the smallest number
Press any key to continue . . .
```

Fig2. Output for a Perl script to accept three number and display smallest number.

Q.3) Write a Perl script to enter two string and check whether its equal or not.

CODE:

```
print "Enter two strings: \n";

$string1 = <stdin>;
```

```

$string2 = <stdin>;
if ($string1 eq $string2)
{
    print "The two strings are equal\n";
}
else
{
    print "The two strings are not equal\n";
}

```

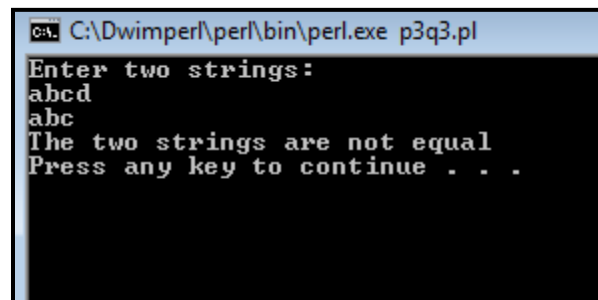


Fig3.1. Output for a Perl script to enter two string and check whether its equal or not.

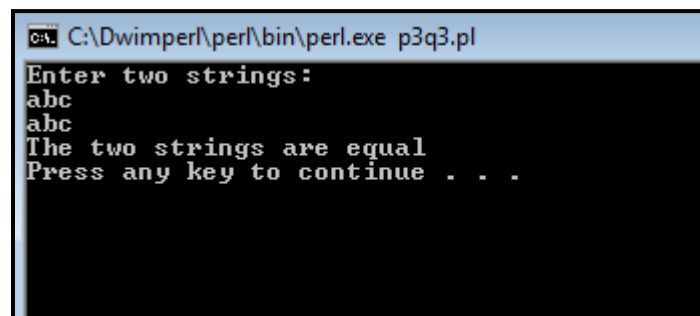


Fig3.2. Output for a Perl script to enter two string and check whether its equal or not.

Q.4) Write a Perl script store elements in an array and perform the following:

- a. Find length of an array
- b. Add one element at end of an array
- c. Remove one element at beginning of an array
- d. Add one element at beginning of an array
- e. Remove one element at end of an array

CODE:

```

@array = ("Pizza","Burger","Chocolate");

$length = @array;

print "a. Length of array is $length\n";

push(@array,"Fries");

print "b. \@array=@array\n";

shift(@array);

print "c. \@array=@array\n";

unshift(@array,"Donut");

print "d. \@array=@array\n";

pop(@array);

print "e. \@array=@array\n";

```

OUTPUT:

```

C:\Dwimper\perl\bin\perl.exe p3q4.pl
a. Length of array is 3
b. @array=Pizza Burger Chocolate Fries
c. @array=Burger Chocolate Fries
d. @array=Donut Burger Chocolate Fries
e. @array=Donut Burger Chocolate
Press any key to continue . . .

```

Fig4. Output for a Perl script store elements in an array and perform length of an array, add one element at end of an array, remove one element at beginning of an array, add one element at beginning of an array, remove one element at end of an array.

Q.5) Write a Perl script create elements in an array like ATGCA, ATTG, AATGC, AAAT and perform the following:

- a. Find length of an array**
- b. Add one element i.e. ATGC at bottom of an array**
- c. Remove one element at beginning of an array**
- d. Add one element i.e. ATGCC at top of an array**
- e. Remove one element at end of an array**

CODE:

```

@array = ("ATGCA","ATTG","AATGC","AAAT");

```



```

$length = @array;
print "a. Length of array is $length\n";
push(@array,"ATGC");
print "b. \@array=@array\n";
shift(@array);
print "c. \@array=@array\n";
unshift(@array,"ATGCC");
print "d. \@array=@array\n";
pop(@array);
print "e. \@array=@array\n";

```

OUTPUT:

```

C:\Dwimperl\perl\bin\perl.exe p3q5.pl
a. Length of array is 4
b. @array=ATGCA ATTG AATGC AAAT ATGC
c. @array=ATTG AATGC AAAT ATGC
d. @array=ATGCC ATTG AATGC AAAT ATGC
e. @array=ATGCC ATTG AATGC AAAT
Press any key to continue . . .

```

Fig5. Output for a Perl script create elements in an array like ATGCA, ATTG, AATGC, AAAT and perform length of an array, add one element i.e. ATGC at bottom of an array, remove one element at beginning of an array, add one element i.e. ATGCC at top of an array, remove one element at end of an array

Q.6) Write a Perl script to create an array and perform following operations such as merge, reverse and sorting.

CODE:

```

@odd = (1,3,5,7,9);
print "@odd\n";

@even = (2,4,6,8);
print "@even\n";

@num = (@odd,@even);
print "After merging two arrays: @num\n";

```

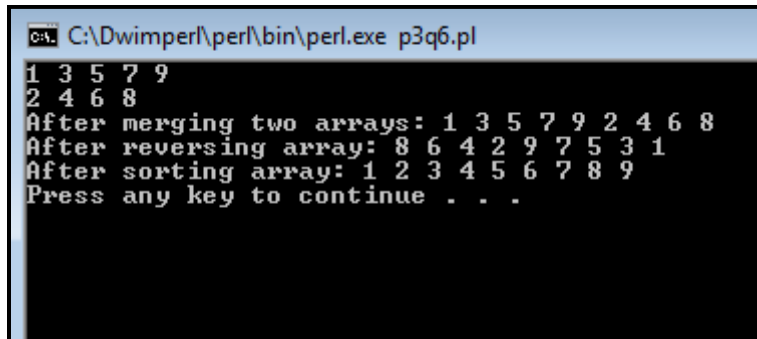
```
@num1 = reverse(@num);

print "After reversing array: @num1\n";

@num2 = sort(@num);

print "After sorting array: @num2\n";
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p3q6.pl
1 3 5 7 9
2 4 6 8
After merging two arrays: 1 3 5 7 9 2 4 6 8
After reversing array: 8 6 4 2 9 7 5 3 1
After sorting array: 1 2 3 4 5 6 7 8 9
Press any key to continue . . .
```

Fig6. Output for a Perl script to create an array and perform following operations such as merge, reverse and sorting.

Q.7) Write a Perl script create an array(numbers) and display in descending order.

CODE:

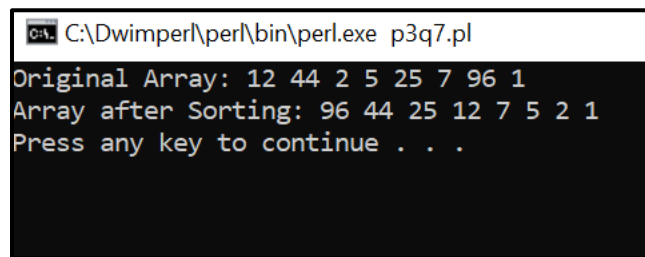
```
@arr = (12, 44, 2, 5, 25, 7, 96, 1);

print "Original Array: @arr\n";

@newarr = sort { $b <=> $a } @arr;

print "Array after Sorting: @newarr\n";
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p3q7.pl
Original Array: 12 44 2 5 25 7 96 1
Array after Sorting: 96 44 25 12 7 5 2 1
Press any key to continue . . .
```

Fig7. Output for perl script create an array(numbers) and display in descending order.

Q.8) Write a Perl script to stored string of an array and display index number 3,4,5 (ELEMENTS)at once.

CODE:

```
@string;
```

```

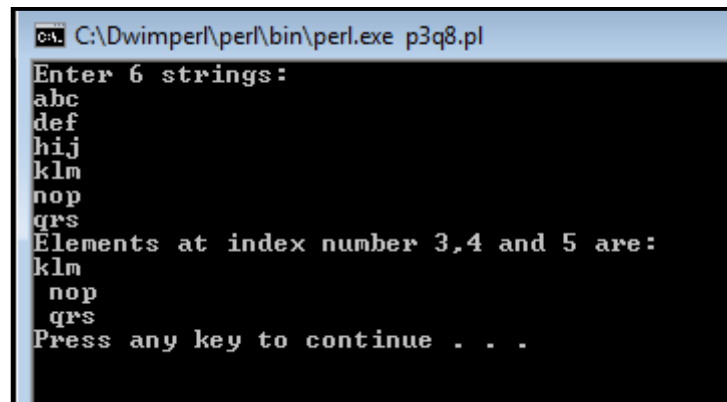
print "Enter 6 strings: \n";
for ($i=0;$i<6;$i++)
{
    $string[$i]=<stdin>;
}

@display=@string[3,4,5];

print "Elements at index number 3,4 and 5 are: \n@display";

```

OUTPUT:



```

C:\Dwimper\perl\bin\perl.exe p3q8.pl
Enter 6 strings:
abc
def
hij
klm
nop
qrs
Elements at index number 3,4 and 5 are:
klm
nop
qrs
Press any key to continue . . .

```

Fig8. Output for a Perl script to stored string of an array and display index number 3,4,5 (ELEMENTS)at once.

Q.9) Write a Perl script to demonstrate splice operator.

CODE:

```

@num=(1,2,3,4,5,6,7,8,9,10);

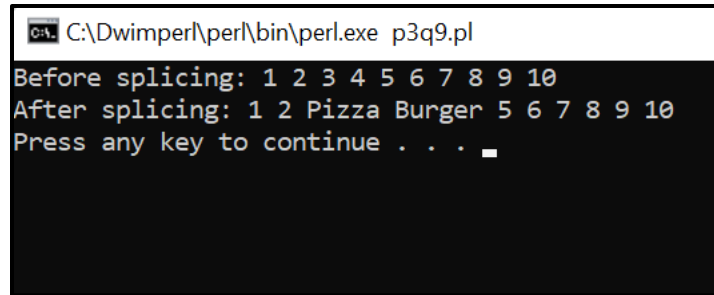
print "Before splicing: @num \n";

splice(@num,2,2,"Pizza", "Burger");

print "After splicing: @num \n";

```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p3q9.pl
Before splicing: 1 2 3 4 5 6 7 8 9 10
After splicing: 1 2 Pizza Burger 5 6 7 8 9 10
Press any key to continue . . .
```

Fig9. Output for a Perl script to demonstrate splice operator.

Q.10) Write a Perl script to sort hashes using keys.

CODE:

```
%data=(1=>"Liza", 2=>"Rucha", 3=>"Vidhi");

print "Before sorting: \n";

print %data;

print "\n";

@data = keys %data;

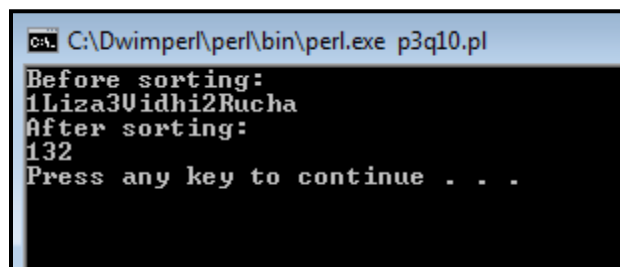
@data = sort({a<=>b} @data);

print "After sorting: \n";

print @data;

print "\n";
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p3q10.pl
Before sorting:
1Liza3Vidhi2Rucha
After sorting:
132
Press any key to continue . . .
```

Fig10. Output for a Perl script to sort hashes using keys.

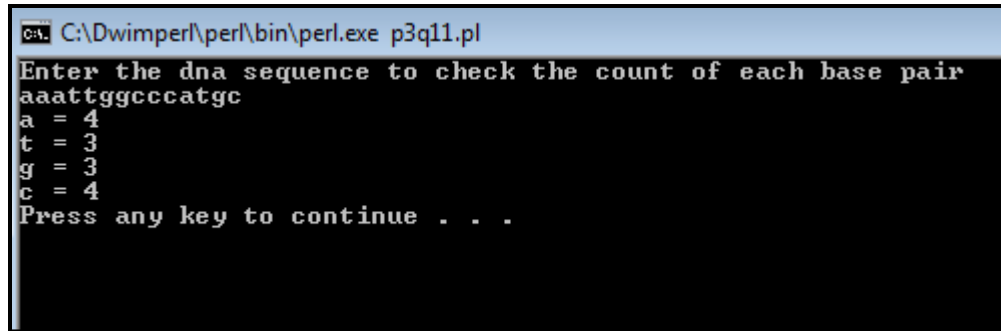
Q.11) Write a Perl program to determine the frequency of nucleotide bases (i.e. A, T, G, and C) in given nucleotide sequence using nested if else.

CODE:

```
print "Enter the dna sequence to check the count of each base pair\n";
```

```
$input = <stdin>;  
chomp($input);  
$a=0;  
$c=0;  
$t=0;  
$g=0;  
for($position=0; $position < length$input; $position ++)  
{  
    $n = substr($input,$position,1);  
    if($n eq "a"){  
        $a++;  
    }elseif($n eq "t"){  
        $t++;  
    }elseif($n eq "g"){  
        $g++;  
    }elseif($n eq "c"){  
        $c++;  
    }  
}  
print"a = $a\n";  
print"t = $t\n";  
print"g = $g\n";  
print"c = $c\n";
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p3q11.pl
Enter the dna sequence to check the count of each base pair
aaattggcccatgc
a = 4
t = 3
g = 3
c = 4
Press any key to continue . . .
```

Fig11. Output for a Perl program to determine the frequency of nucleotide bases (i.e. A, T, G, and C) in given nucleotide sequence using nested if else

Practical No 4

Subroutine

AIM:

To understand and write program for perl subroutine

THEORY:

A Perl subroutine or function is a group of statements that together performs a task. You can divide up your code into separate subroutines. How you divide up your code among different subroutines is up to you, but logically the division usually is so each function performs a specific task.

Perl uses the terms subroutine, method and function interchangeably.

Define and Call a Subroutine

The general form of a subroutine definition in Perl programming language is as follows –

```
sub subroutine_name {  
  body of the subroutine  
}
```

The typical way of calling that Perl subroutine is as follows –

```
subroutine_name( list of arguments );
```

Passing Arguments to a Subroutine

You can pass various arguments to a subroutine like you do in any other programming language and they can be accessed inside the function using the special array `@_`. Thus the first argument to the function is in `$_[0]`, the second is in `$_[1]`, and so on.

You can pass arrays and hashes as arguments like any scalar

Q.1) Write a Perl program to Create a subroutine named Calculate and find area and perimeter a rectangle.

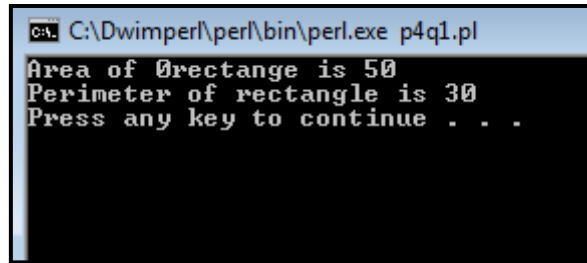
CODE:

```
sub calculate  
{  
    $l = 10;  
    $b = 5;  
    $area = $l*$b;  
    $perimeter = 2*($l+$b);  
    print "Area of Rectangle is $area\n";  
}
```

```

    print "Perimeter of rectangle is $perimeter\n";
}
calculate();

```



```

C:\Dwimper\perl\bin\perl.exe p4q1.pl
Area of 0rectangle is 50
Perimeter of rectangle is 30
Press any key to continue . . .

```

Fig1. Output for a Perl program to Create a subroutine named Calculate and find area and perimeter a rectangle

Q.2) Write a Perl program to Create a subroutine named Calculate and find area and perimeter a rectangle with parameter.

CODE:

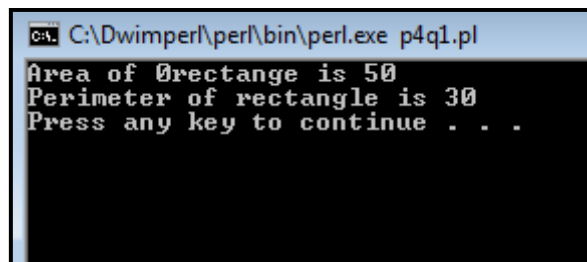
```

sub calculate
{
    my ($l,$b)=@_;
    $area = $l*$b;
    $perimeter = 2*($l+$b);
    print "Area of 0rectangle is $area\n";
    print "Perimeter of rectangle is $perimeter\n";
}

$l = 10;
$b = 5;
calculate($l,$b);

```

OUTPUT:



```

C:\Dwimper\perl\bin\perl.exe p4q1.pl
Area of 0rectangle is 50
Perimeter of rectangle is 30
Press any key to continue . . .

```


Fig2. Output for a Perl program to Create a subroutine named Calculate and find area and perimeter a rectangle with parameter

DATE: 01/08/22

Practical No 5

References and Dereferences, and Scope of variables

AIM:

To understand and write perl program for references and dereferences, and scope of variables

THEORY:

A Perl reference is a scalar data type that holds the location of another value which could be scalar, arrays, or hashes. Because of its scalar nature, a reference can be used anywhere, a scalar can be used.

You can construct lists containing references to other lists, which can contain references to hashes, and so on. This is how the nested data structures are built in Perl.

Create References

It is easy to create a reference for any variable, subroutine or value by prefixing it with a backslash as follows –

```
$scalarref = \$foo;  
$arrayref = \@ARGV;  
$hashref = \%ENV;  
$coderef = \&handler;  
$globref = \*foo;
```

Dereferencing

Dereferencing returns the value from a reference point to the location. To dereference a reference simply use \$, @ or % as prefix of the reference variable depending on whether the reference is pointing to a scalar, array, or hash.

References to Functions

This might happen if you need to create a signal handler so you can produce a reference to a function by preceding that function name with \& and to dereference that reference you simply need to prefix reference variable using ampersand &.

Create a reference to above function.

```
$cref = \&PrintHash;
```

Function call using reference.

```
&$cref(%hash);
```

Private Variables in a Subroutine

By default, all variables in Perl are **global variables**, which means they can be accessed from anywhere in the program. But you can create **private** variables called **lexical variables** at any time with the **my** operator.

The **my** operator confines a variable to a particular region of code in which it can be used and accessed. Outside that region, this variable cannot be used or accessed. This region is called its scope. A lexical scope is usually a block of code with a set of braces around it, such as those defining the body of the subroutine or those marking the code blocks of *if*, *while*, *for*, *foreach*, and *eval* statements.

```
sub somefunc {  
    my $variable; # $variable is invisible outside somefunc()  
    my ($another, @an_array, %a_hash); # declaring many variables at once  
}
```

State Variables via state()

There are another type of lexical variables, which are similar to private variables but they maintain their state and they do not get reinitialized upon multiple calls of the subroutines. These variables are defined using the **state** operator

```
state $count = 0;
```

Q.1) Write a Perl script to accept a number and create reference of scalar variable and display a value using dereferencing.

CODE:

```
print "Enter a number: \n";  
  
$n = <stdin>;  
  
$nref = \$n;  
  
$a=${$nref};  
  
print "The number is $a \n";
```

OUTPUT:

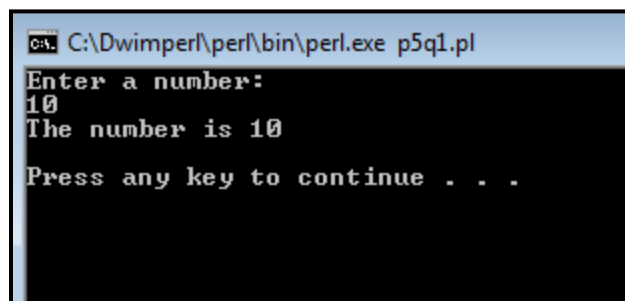


Fig1. Output for a Perl script to accept a number and create reference of scalar variable and display a value using dereferencing

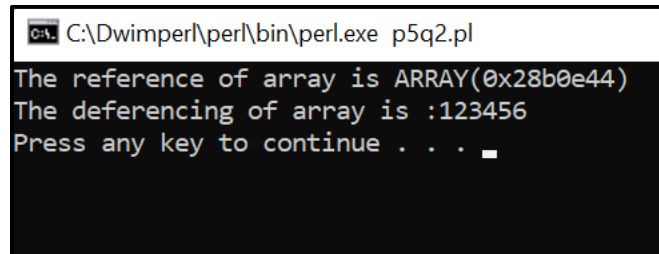
Q.2) Write a Perl script to store an array and use reference and dereferencing.

CODE:

```
@array = (1,2,3,4,5,6);  
  
$ref = \@array;
```

```
print "The reference of array is $ref\n";  
print "The deferencing of array is :";  
print @ $ref;  
print "\n";
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p5q2.pl  
The reference of array is ARRAY(0x28b0e44)  
The deferencing of array is :123456  
Press any key to continue . . .
```

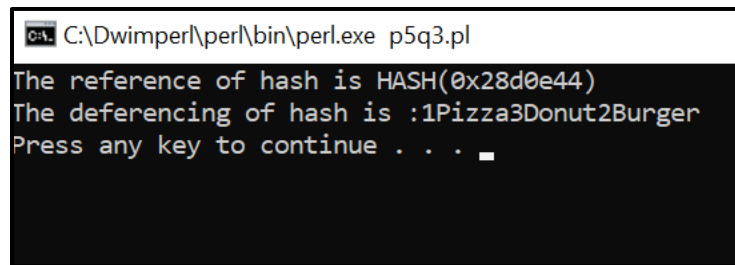
Fig2. Output for a Perl script to store an array and use reference and dereferencing

Q.3) Write a Perl script to store a hash and use reference and dereferencing.

CODE:

```
%hash = (1=>"Pizza",2=>"Burger",3=>"Donut");  
$ref = \%hash;  
print "The reference of hash is $ref\n";  
print "The deferencing of hash is :";  
print % $ref;  
print "\n";
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p5q3.pl  
The reference of hash is HASH(0x28d0e44)  
The deferencing of hash is :1Pizza3Donut2Burger  
Press any key to continue . . .
```

Fig3. Output for a Perl script to store a hash and use reference and dereferencing

Q.4) Write a Perl script to create a subroutine and use reference and dereferencing.

CODE:

```
sub array
```

```

{
    @arr=(1,2,3,4);
}
$ref=\&array;
print "The deferencing of function is :";
print & $ref;
print "\n";

```

OUTPUT:

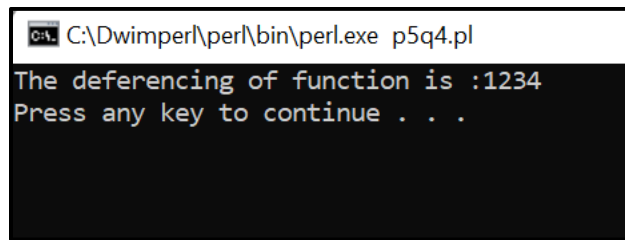


Fig4. Output for a Perl script to create a subroutine and use reference and dereferencing

Q.5) Write a Perl script to store a number in global variable and demonstrate the scope of it.

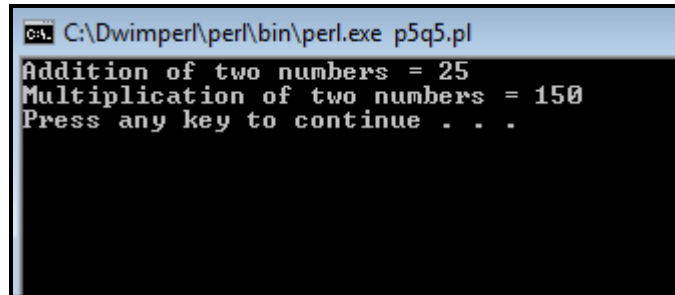
CODE:

```

sub add
{
    $a=10;
    $b=15;
    $c=$a+$b;
    print "Addition of two numbers = $c\n";
}
add();
$c=$a*$b;
print "Multiplication of two numbers = $c\n";

```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimper\perl\bin\perl.exe p5q5.pl'. The output text is: 'Addition of two numbers = 25', 'Multiplication of two numbers = 150', and 'Press any key to continue . . .'.

```
C:\Dwimper\perl\bin\perl.exe p5q5.pl
Addition of two numbers = 25
Multiplication of two numbers = 150
Press any key to continue . . .
```

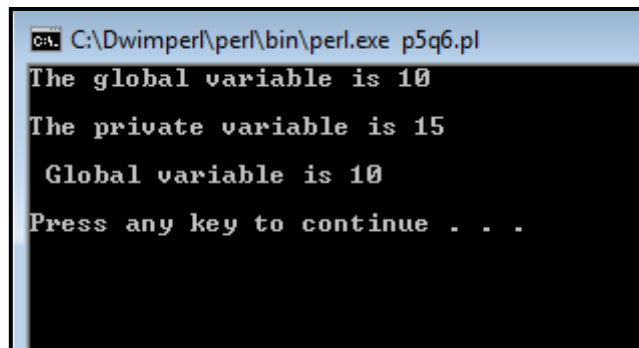
Fig5. Output for a Perl script to store a number in global variable and demonstrate the scope of it.

Q.6) Write a Perl script to store a number in private variable and demonstrate the scope of it.

CODE:

```
$num="10\n";
print"The global variable is $num\n";
sub globn {
my $num = "15\n";
print "The private variable is $num\n";
}
globn();
print " Global variable is $num\n";
```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimper\perl\bin\perl.exe p5q6.pl'. The output text is: 'The global variable is 10', 'The private variable is 15', 'Global variable is 10', and 'Press any key to continue . . .'.

```
C:\Dwimper\perl\bin\perl.exe p5q6.pl
The global variable is 10
The private variable is 15
Global variable is 10
Press any key to continue . . .
```

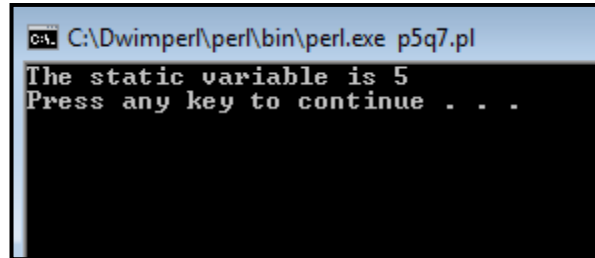
Fig6. Output for a Perl script to store a number in private variable and demonstrate the scope of it.

Q.7) Write a Perl script to store a number in state variable and display an output.

CODE:

```
use feature "state";  
state $num = "5";  
print "The static variable is $num\n";
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p5q7.pl  
The static variable is 5  
Press any key to continue . . .
```

Fig7. Output for a Perl script to store a number in state variable and display an output.

Practical No.6

Regular Expressions

AIM:

To understand and write perl programs for regular expressions.

THEORY:

A regular expression is a string of characters that defines the pattern or patterns you are viewing. The syntax of regular expressions in Perl is very similar to what you will find within other regular expression.supporting programs, such as **sed**, **grep**, and **awk**.

The basic method for applying a regular expression is to use the pattern binding operators `=~` and `!~`. The first operator is a test and assignment operator.

There are three regular expression operators within Perl.

- Match Regular Expression - `m//`
- Substitute Regular Expression - `s//`
- Transliterate Regular Expression - `tr//`

The forward slashes in each case act as delimiters for the regular expression (regex) that you are specifying. If you are comfortable with any other delimiter, then you can use in place of forward slash.

The Match Operator

The match operator, `m//`, is used to match a string or statement to a regular expression. For example, to match the character sequence "foo" against the scalar `$bar`, you might use a statement like this –

```
$bar =~ /foo/
```

The `m//` actually works in the same fashion as the `q//` operator series.you can use any combination of naturally matching characters to act as delimiters for the expression. For example, `m{ }`, `m()`, and `m><` are all valid.

The Substitution Operator

The substitution operator, `s//`, is really just an extension of the match operator that allows you to replace the text matched with some new text. The basic form of the operator is –

```
s/PATTERN/REPLACEMENT/;
```

The **PATTERN** is the regular expression for the text that we are looking for. The **REPLACEMENT** is a specification for the text or regular expression that we want to use to replace the found text with. For example, we can replace all occurrences of **dog** with **cat** using the following regular expression –

```
$string =~ s/cat/dog/;
```

Match and Substitution Operator Modifiers

The match and substitution operator supports its own set of modifiers. The /g modifier allows for global matching. The /i modifier will make the match case insensitive. Here is the complete list of modifiers

Sr.No.	Modifier & Description
1	i Makes the match case insensitive.
2	m Specifies that if the string has newline or carriage return characters, the ^ and \$ operators will now match against a newline boundary, instead of a string boundary.
3	o Evaluates the expression only once.
4	s Allows use of . to match a newline character.
5	x Allows you to use white space in the expression for clarity.
6	g Globally finds all matches.
7	cg Allows the search to continue even after a global match fails.

The Translation Operator

Translation is similar, but not identical, to the principles of substitution, but unlike substitution, translation (or transliteration) does not use regular expressions for its search on replacement values. The translation operators are –

tr/SEARCHLIST/REPLACEMENTLIST/

y/SEARCHLIST/REPLACEMENTLIST/

The translation replaces all occurrences of the characters in SEARCHLIST with the corresponding characters in REPLACEMENTLIST. For example, using the "The cat sat on the mat."

```
$string = 'The cat sat on the mat';
```

```
$string =~ tr/a/o/;
```

Translation Operator Modifiers

Following is the list of operators related to translation.

Sr.No.	Modifier & Description
1	c Complements SEARCHLIST.
2	d Deletes found but unreplaced characters.
3	S Squashes duplicate replaced characters.

Q.1) Write a Perl to accept a DNA sequence and match against pattern “aatg” is found on entered sequence and check whether the match case insensitive.

CODE:

```
print "Enter DNA sequence: \n";
```

```
$dna = <stdin>;
```

```
if ($dna =~ m/aatg/i)
```

```
{
```

```
    print "aatg is present in the sequence\n";
```

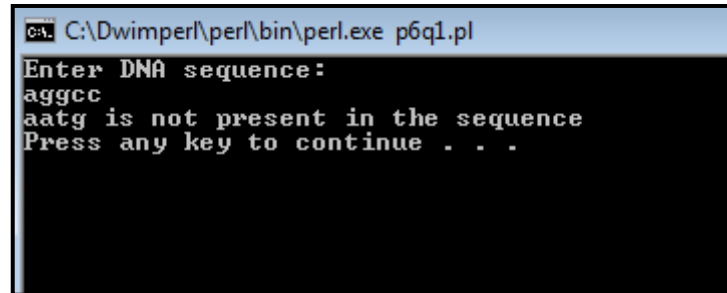
```
}
```

```
else
```

```
{
```

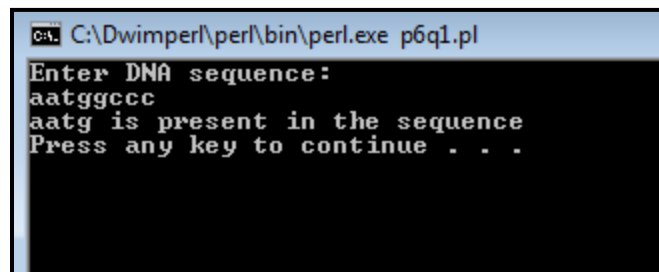
```
print "aatg is not present in the sequence\n";  
}
```

OUTPUT:



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimper\perl\bin\perl.exe p6q1.pl'. The prompt 'Enter DNA sequence:' is followed by the user input 'aggcc'. The output shows 'aatg is not present in the sequence' followed by 'Press any key to continue . . .'. The window has a black background with white text.

Fig1.1 Output for Perl to accept a DNA sequence and match against pattern “aatg”



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimper\perl\bin\perl.exe p6q1.pl'. The prompt 'Enter DNA sequence:' is followed by the user input 'aatggccc'. The output shows 'aatg is present in the sequence' followed by 'Press any key to continue . . .'. The window has a black background with white text.

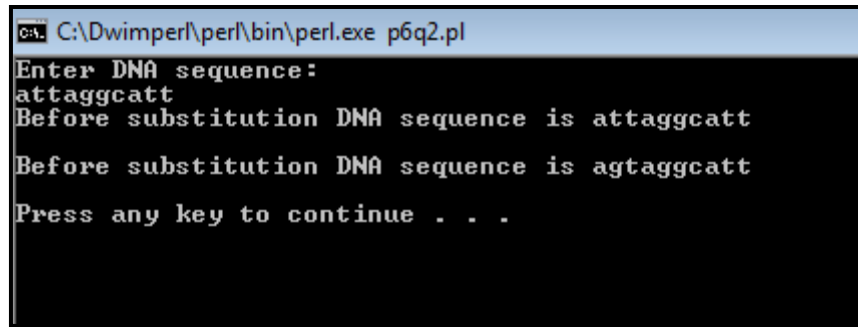
Fig1.2 Output for Perl to accept a DNA sequence and match against pattern “aatg”

Q.2) Write a Perl script to accept DNA sequence from user and search pattern “att” and replace with “agt”.

CODE:

```
print "Enter DNA sequence: \n";  
  
$dna = <stdin>;  
  
print "After replacement DNA sequence is $dna \n";  
  
$dna=~s/att/agt/;  
  
print "After replacement DNA sequence is $dna \n";
```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimperl\perl\bin\perl.exe p6q2.pl'. The prompt is 'C:\>'. The user enters 'attaggcatt' in response to 'Enter DNA sequence:'. The script then prints 'Before substitution DNA sequence is attaggcatt' and 'Before substitution DNA sequence is agtaggcatt' (note the typo 'ag' instead of 'att'). It ends with 'Press any key to continue . . .'.

```
C:\> C:\Dwimperl\perl\bin\perl.exe p6q2.pl
Enter DNA sequence:
attaggcatt
Before substitution DNA sequence is attaggcatt
Before substitution DNA sequence is agtaggcatt
Press any key to continue . . .
```

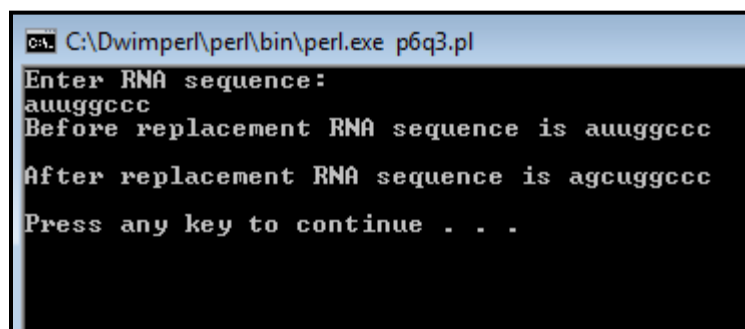
Fig2. Output for Perl script to accept DNA sequence from user and search pattern “att” and replace with “agt”

Q.3) Write a Perl script to accept RNA sequence from user and search pattern “auu” and replace with “agcu” in whole sequence.

CODE:

```
print "Enter RNA sequence: \n";
$rna = <stdin>;
print "Before replacement RNA sequence is $rna \n";
$rna=~s/auu/agcu/;
print "After replacement RNA sequence is $rna \n";
```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimperl\perl\bin\perl.exe p6q3.pl'. The prompt is 'C:\>'. The user enters 'auuggccc' in response to 'Enter RNA sequence:'. The script then prints 'Before replacement RNA sequence is auuggccc' and 'After replacement RNA sequence is agcuggccc' (note the typo 'agcu' instead of 'auu'). It ends with 'Press any key to continue . . .'.

```
C:\> C:\Dwimperl\perl\bin\perl.exe p6q3.pl
Enter RNA sequence:
auuggccc
Before replacement RNA sequence is auuggccc
After replacement RNA sequence is agcuggccc
Press any key to continue . . .
```

Fig3. Output for Perl script to accept RNA sequence from user and search pattern “auu” and replace with “agcu” in whole sequence

Q.4) Write a Perl script to accept RNA sequence and convert that into DNA sequence.

CODE:

```
print "Enter RNA sequence: \n";
$rna = <stdin>;
```

```
print "The RNA sequence is $rna \n";
```

```
$rna=~tr/u/t/;
```

```
print "The DNA sequence is $rna \n";
```

OUTPUT:

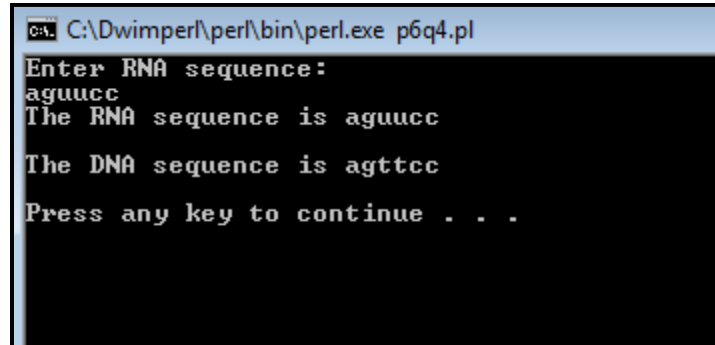
A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimper\perl\bin\perl.exe p6q4.pl'. The prompt is 'C:\>'. The user enters 'Enter RNA sequence:' followed by 'aguucc'. The script outputs 'The RNA sequence is aguucc'. Then it outputs 'The DNA sequence is agttcc'. Finally, it outputs 'Press any key to continue . . .'.

Fig4. Output for Perl script to accept RNA sequence and convert that into DNA sequence

Q.5) Write a Perl script to accept a string and remove duplicate characters from entered string.

CODE:

```
print "Enter a string: \n";
```

```
$str = <stdin>;
```

```
print "Before removing duplicate characters: $str\n";
```

```
$str =~ tr/a-z/a-z/s;
```

```
print "After removing duplicate characters: $str\n";
```

OUTPUT:

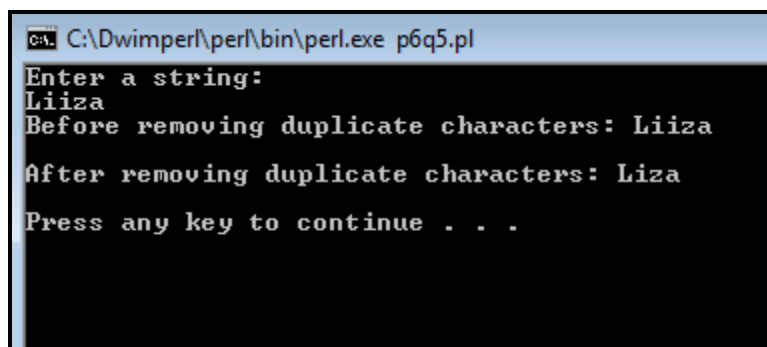
A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimper\perl\bin\perl.exe p6q5.pl'. The prompt is 'C:\>'. The user enters 'Enter a string:' followed by 'Liiza'. The script outputs 'Before removing duplicate characters: Liiza'. Then it outputs 'After removing duplicate characters: Liza'. Finally, it outputs 'Press any key to continue . . .'.

Fig5. Output for Perl script to accept a string and remove duplicate characters from entered string

Practical No. 7**Metacharacters, Quantifiers and Substring****AIM:**

To understand and write Perl programs for Metacharacters, Quantifiers and Substring.

THEORY:**Perl Metacharacters:**

Sr.No.	Pattern & Description
1	^ Matches beginning of line.
2	\$ Matches end of line.
3	. Matches any single character except newline. Using m option allows it to match newline as well.
4	* Matches 0 or more occurrences of preceding expression.
5	+ Matches 1 or more occurrence of preceding expression.
6	? Matches 0 or 1 occurrence of preceding expression.
7	{ n } Matches exactly n number of occurrences of preceding expression.

8	a b Matches either a or b.
9	() Used for grouping
10	[] Matches set of characters
11	\ Matches special characters

Perl Quantifiers:

* Matches 0 or more occurrences of preceding expression.
+ Matches 1 or more occurrence of preceding expression.
? Matches 0 or 1 occurrence of preceding expression.

Perl SubString function:

This function returns a substring of EXPR, starting at OFFSET within the string. If OFFSET is negative, starts that many characters from the end of the string. If LEN is specified, returns that number of bytes, or all bytes up until end-of-string if not specified. If LEN is negative, leaves that many characters off the end of the string.

If REPLACEMENT is specified, replaces the substring with the REPLACEMENT string.

If you specify a substring that passes beyond the end of the string, it returns only the valid element of the original string.

Syntax

Following is the simple syntax for this function –

substr EXPR, OFFSET, LEN, REPLACEMENT

substr EXPR, OFFSET, LEN

substr EXPR, OFFSET

Return Value

This function returns string.

Example

Following is the example code showing its basic usage –

```
$temp = substr("okay", 2);  
print "Substring valuye is $temp\n";  
$temp = substr("okay", 1,2);  
print "Substring valuye is $temp\n";
```

Q.1) Write a Perl program to check that a string contains only a certain set of characters (in this case a-z, A-Z and 0-9).

CODE:

```
print "Enter a string: \n";  
$str = <stdin>;  
if ($str =~m/[a-zA-Z0-9]/g)  
{  
    print "String contains only word character\n"  
}  
else  
{  
    print "String does not contain only word character\n"  
}
```

OUTPUT:

```
C:\Dwimperl\perl\bin\perl.exe p7q1.pl
Enter a string:
////////
String does not contain only word character
Press any key to continue . . .
```

Fig1.1 Output for Perl program to check that a string contains only a certain set of characters

```
C:\Dwimperl\perl\bin\perl.exe p7q1.pl
Enter a string:
abcd09
String contains only word character
Press any key to continue . . .
```

Fig1.2 Output for Perl program to check that a string contains only a certain set of characters

Q.2) Write a Perl program that matches a string that has an a followed by zero or more t's in a DNA sequence.

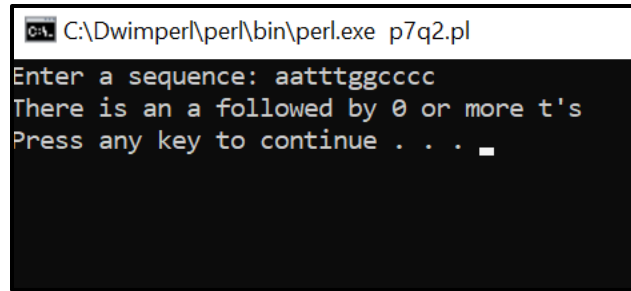
CODE:

```
print("Enter a sequence: ");

$seq = <stdin>;

if($seq =~ m/at*/){
    print("There is an a followed by 0 or more t's\n");
}else{
    print("There is not an a followed by 0 or more t's\n");
}
```


OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p7q2.pl
Enter a sequence: aatttggcccc
There is an a followed by 0 or more t's
Press any key to continue . . .
```

Fig 2. Output for Perl program that matches a string that has an a followed by zero or more t's in a DNA sequence

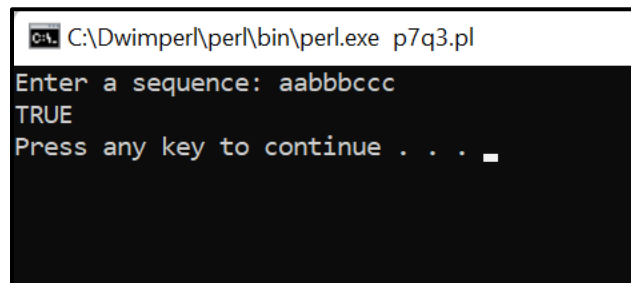
Q.3) Write a Perl program that matches a string that has an a followed by one or more b's.

CODE:

```
print("Enter a sequence: ");
$string = <stdin>;

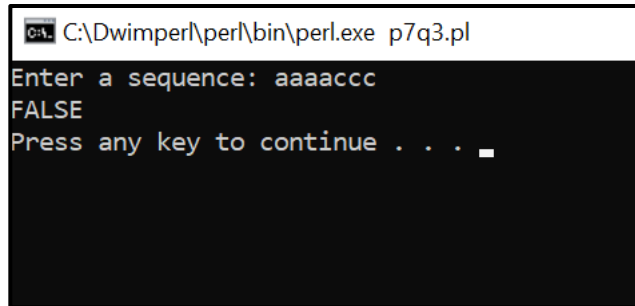
if($string =~ m/ab+){
    print("TRUE\n");
}else{
    print("FALSE\n");
}
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p7q3.pl
Enter a sequence: aabbbccc
TRUE
Press any key to continue . . .
```

Fig3.1. Output for Perl program that matches a string that has an a followed by one or more b's

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimperl\perl\bin\perl.exe p7q3.pl'. The command prompt displays the following text: 'Enter a sequence: aaaaccc', 'FALSE', and 'Press any key to continue . . .'.

```
C:\Dwimperl\perl\bin\perl.exe p7q3.pl
Enter a sequence: aaaaccc
FALSE
Press any key to continue . . .
```

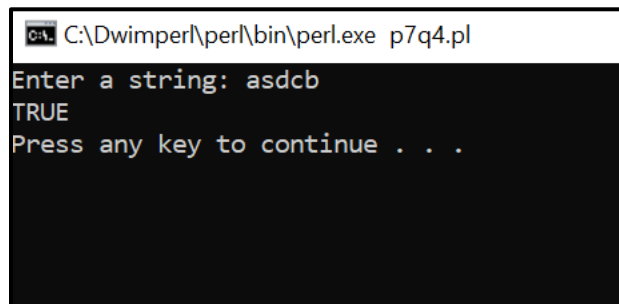
Fig3.2. Output for Perl program that matches a string that has an a followed by one or more b's

Q.4) Write a Perl program that matches a string that has an “a” followed by anything, ending in “b”.

CODE:

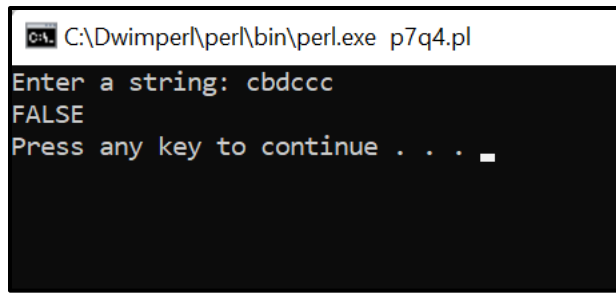
```
print("Enter a string: ");
$string = <stdin>;
if($string =~ m/a/ && $string =~ m/b$/){
    print("TRUE\n");
}else{
    print("FALSE\n");
}
```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimperl\perl\bin\perl.exe p7q4.pl'. The command prompt displays the following text: 'Enter a string: asdcb', 'TRUE', and 'Press any key to continue . . .'.

```
C:\Dwimperl\perl\bin\perl.exe p7q4.pl
Enter a string: asdcb
TRUE
Press any key to continue . . .
```

Fig4.1. Output for Perl program that matches a string that has an “a” followed by anything, ending in “b”

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimperl\perl\bin\perl.exe p7q4.pl'. The prompt 'Enter a string:' is followed by the input 'cbdccc'. The output is 'FALSE'. Below that, it says 'Press any key to continue . . .'.

```
C:\Dwimperl\perl\bin\perl.exe p7q4.pl
Enter a string: cbdccc
FALSE
Press any key to continue . . .
```

Fig4.1. Output for Perl program that matches a string that has an “a” followed by anything, ending in “b”

Q.5) Write a Perl program that matches a word at the beginning of a string.

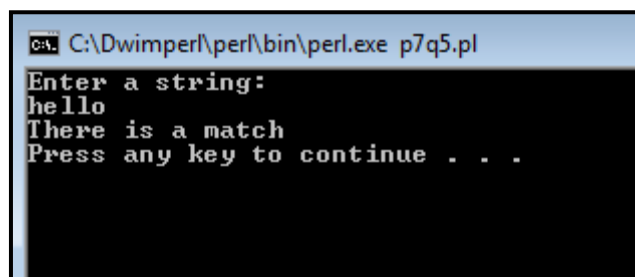
CODE:

```
print "Enter a string: \n";

$str = <stdin>;

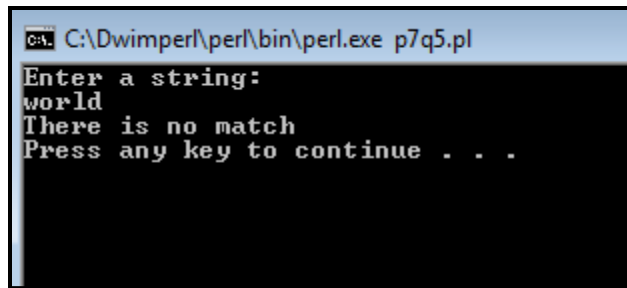
if ($str =~m/^hello/)
{
    print "There is a match\n"
}
else
{
    print "There is no match\n"
}
```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimperl\perl\bin\perl.exe p7q5.pl'. The prompt 'Enter a string:' is followed by the input 'hello'. The output is 'There is a match'. Below that, it says 'Press any key to continue . . .'.

```
C:\Dwimperl\perl\bin\perl.exe p7q5.pl
Enter a string:
hello
There is a match
Press any key to continue . . .
```

Fig 5.1. Output for Perl program that matches a word at the beginning of a string.



```
C:\Dwimperl\perl\bin\perl.exe p7q5.pl
Enter a string:
world
There is no match
Press any key to continue . . .
```

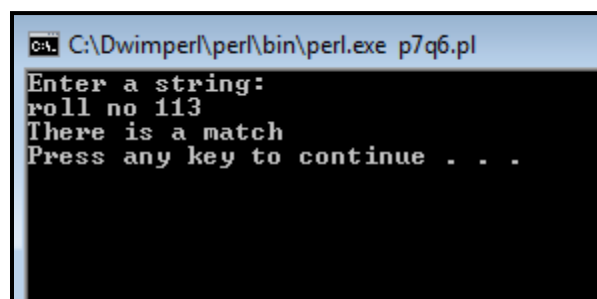
Fig 5.2. Output for Perl program that matches a word at the beginning of a string.

Q.6) Write a Perl program to check for a number at the end of a string.

CODE:

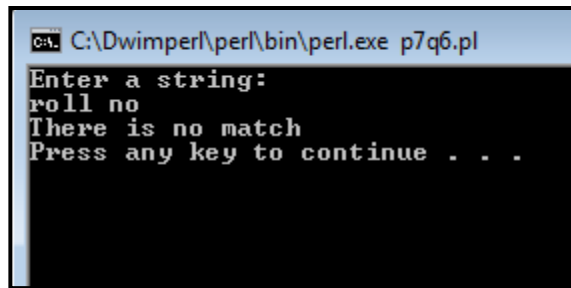
```
print "Enter a string: \n";
$str = <stdin>;
if ($str =~ m/[0-9]$/)
{
    print "There is a match\n"
}
else
{
    print "There is no match\n"
}
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p7q6.pl
Enter a string:
roll no 113
There is a match
Press any key to continue . . .
```

Fig 6.1. Output for Perl program to check for a number at the end of a string.

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimper\perl\bin\perl.exe p7q6.pl'. The command prompt displays the following text: 'Enter a string:', 'roll no', 'There is no match', and 'Press any key to continue . . .'.

```
C:\Dwimper\perl\bin\perl.exe p7q6.pl
Enter a string:
roll no
There is no match
Press any key to continue . . .
```

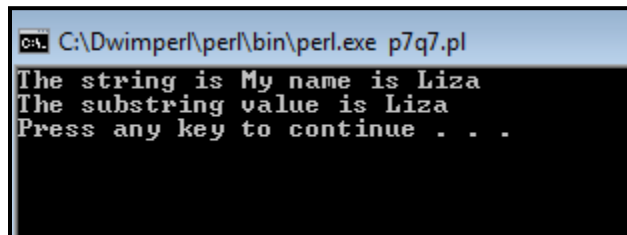
Fig 6.2. Output for Perl program to check for a number at the end of a string.

Q.7) Write a Perl program to demonstrate substr() function.

CODE:

```
$str = "My name is Liza";
print "The string is $str\n";
$str1 = substr($str, 11, 4);
print "The substring value is $str1\n";
```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimper\perl\bin\perl.exe p7q7.pl'. The command prompt displays the following text: 'The string is My name is Liza', 'The substring value is Liza', and 'Press any key to continue . . .'.

```
C:\Dwimper\perl\bin\perl.exe p7q7.pl
The string is My name is Liza
The substring value is Liza
Press any key to continue . . .
```

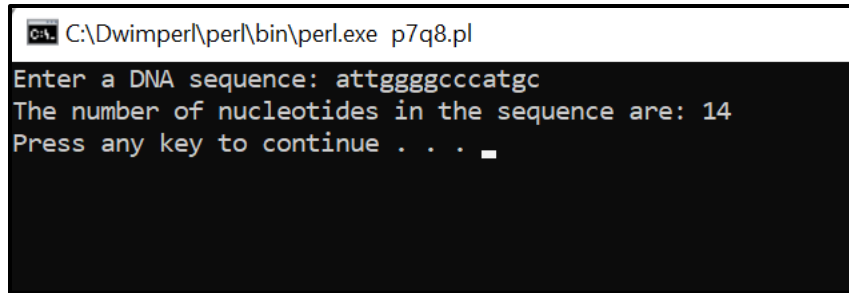
Fig 7. Output for Perl program to demonstrate substr() function

Q.8) Write a Perl program to count number of nucleotides in a sequence using regex.

CODE:

```
print("Enter a DNA sequence: ");
$string = <stdin>;
print("The number of nucleotides in the sequence are: ");
print($string =~ s/\w/\w/g);
print("\n");
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p7q8.pl
Enter a DNA sequence: attggggcccatgc
The number of nucleotides in the sequence are: 14
Press any key to continue . . .
```

Fig8. Output for Perl program to count number of nucleotides in a sequence using regex

Q.9) Write a Perl program accept a string from user and convert into upper case, lower case and display length.

CODE:

```
use 5.010;

print "Enter a string: \n";

$str = <stdin>;

print "The string is upper case is \n";

say uc $str;

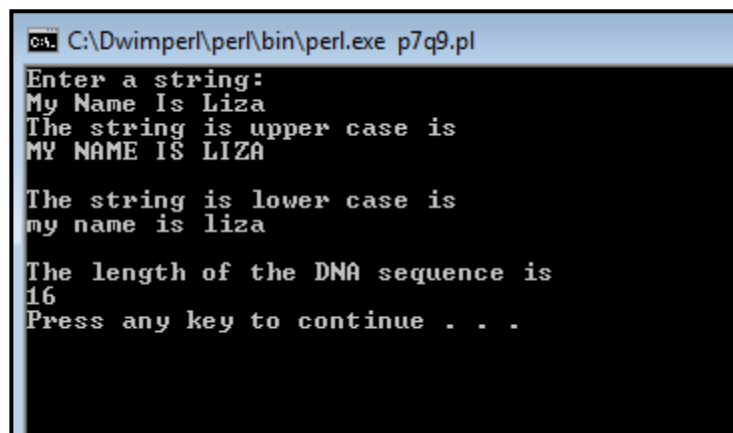
print "The string is lower case is \n";

say lc $str;

print "The length of the DNA sequence is \n";

say length $str;
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p7q9.pl
Enter a string:
My Name Is Liza
The string is upper case is
MY NAME IS LIZA

The string is lower case is
my name is liza

The length of the DNA sequence is
16
Press any key to continue . . .
```

Fig9. Output for Perl program accept a string from user and convert into upper case, lower case and display length

Q.10) Write a Perl program accept an RNA sequence and display the index number of an entered character.

CODE:

```
use 5.010;

print "Enter a RNA sequence: \n";

$rna = <stdin>;

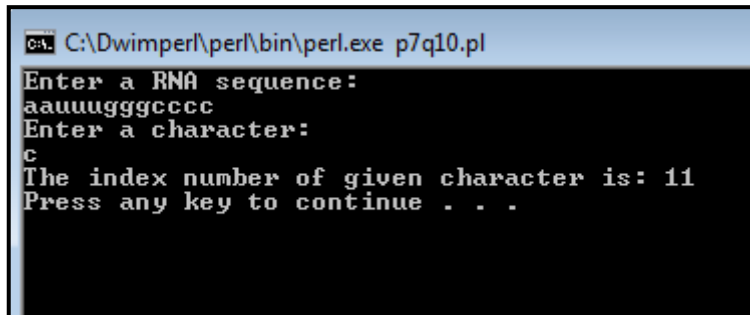
print "Enter a character: \n";

$car = <stdin>;

print "The index number of given character is: ";

say index $rna, $car;
```

OUTPUT:



```
CA: C:\Dwimperl\perl\bin\perl.exe p7q10.pl
Enter a RNA sequence:
aaauuggggcccc
Enter a character:
c
The index number of given character is: 11
Press any key to continue . . .
```

Fig10. Output for Perl program accept an RNA sequence and display the index number of an entered character

Q.11) Write a Perl program to match pattern t or u from an entered sequence.

CODE:

```
print "Enter DNA or RNA sequence: \n";

$seq = <stdin>;

if ($seq =~ m/[t|u]/)

{

    print "It is a match \n";

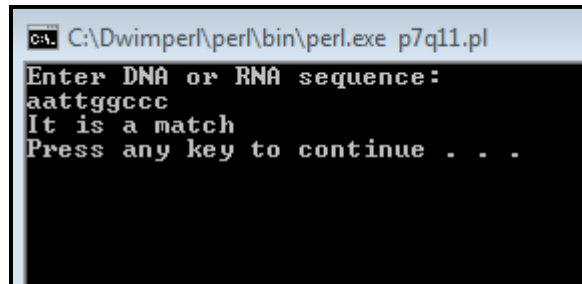
}
```

```

else
{
    print "It is not a match \n";
}

```

OUTPUT:

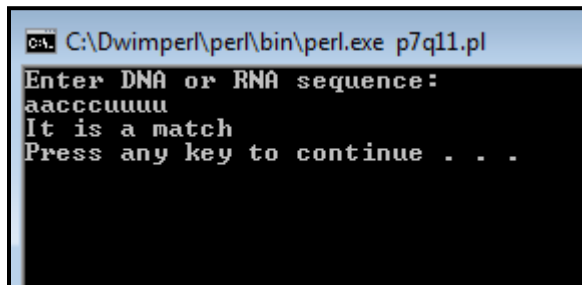


```

C:\Dwimper\perl\bin\perl.exe p7q11.pl
Enter DNA or RNA sequence:
aattggccc
It is a match
Press any key to continue . . .

```

Fig11.1. Output for Perl program to match pattern t or u from an entered sequence

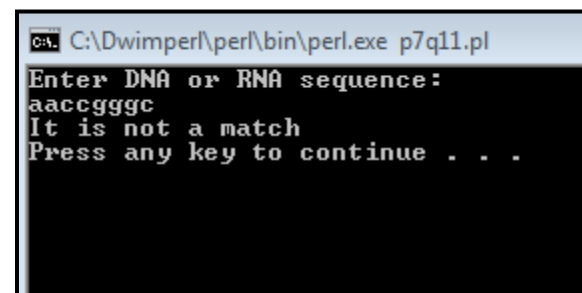


```

C:\Dwimper\perl\bin\perl.exe p7q11.pl
Enter DNA or RNA sequence:
aaccuuuuu
It is a match
Press any key to continue . . .

```

Fig11.2. Output for Perl program to match pattern t or u from an entered sequence



```

C:\Dwimper\perl\bin\perl.exe p7q11.pl
Enter DNA or RNA sequence:
aaccgggc
It is not a match
Press any key to continue . . .

```

Fig11.3. Output for Perl program to match pattern t or u from an entered sequence

Q.12) Write a Perl script to split on character “#”.

CODE:

```

print "Enter a string: \n";

$str = <stdin>;

@array = split ('#', $str);

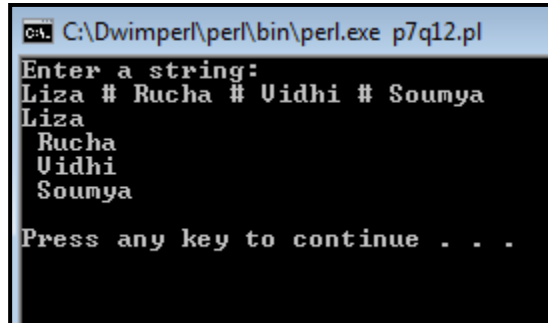
foreach $n (@array)

```



```
{
    print "$n \n";
}
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p7q12.pl
Enter a string:
Liza # Rucha # Uidhi # Soumya
Liza
Rucha
Uidhi
Soumya
Press any key to continue . . .
```

Fig12. Output for Perl script to split on character “#”

Q.13) Write a Perl script to split on pattern i.e. whitespaces.

CODE:

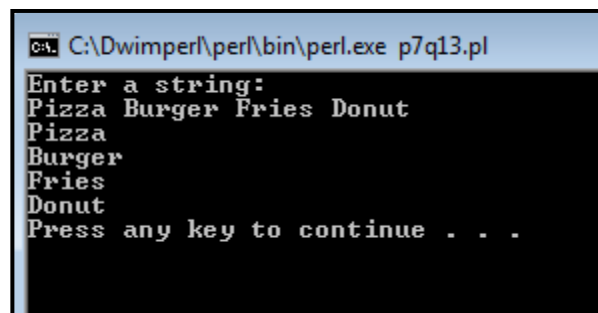
```
print "Enter a string: \n";

$str = <stdin>;

@array = split ( /\s/, $str);

foreach $n (@array)
{
    print "$n \n";
}
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p7q13.pl
Enter a string:
Pizza Burger Fries Donut
Pizza
Burger
Fries
Donut
Press any key to continue . . .
```

Fig13. Output for Perl script to split on pattern i.e. whitespaces

Q.14) Write a Perl script to split on digit and join using single comma.

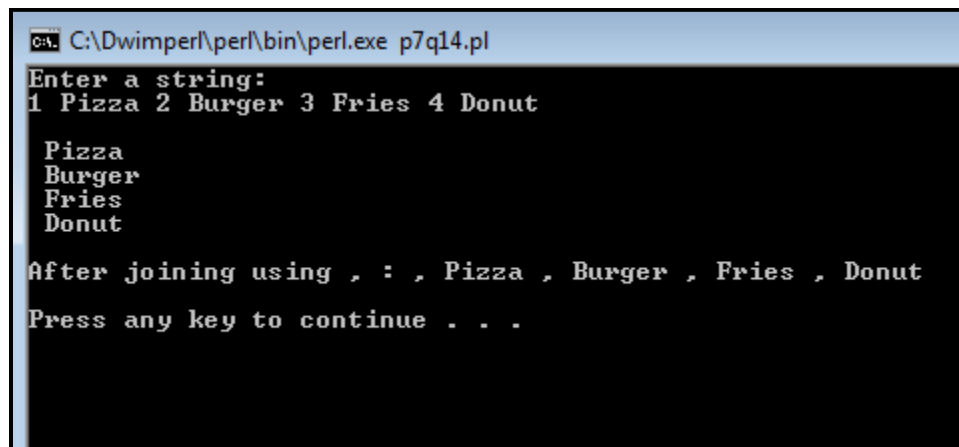
CODE:

```

print "Enter a string: \n";
$str = <stdin>;
@array = split ( /\d/, $str);
foreach $n (@array)
{
    print "$n \n";
}
$str1 = join (",", @array);
print "After joining using , : $str1 \n";

```

OUTPUT:



```

C:\Dwimper\perl\bin\perl.exe p7q14.pl
Enter a string:
1 Pizza 2 Burger 3 Fries 4 Donut

Pizza
Burger
Fries
Donut

After joining using , : , Pizza , Burger , Fries , Donut
Press any key to continue . . .

```

Fig14. Output for Perl script to split on digit and join using single comma

Q.15) Write a Perl script to demonstrate on splitting on string.

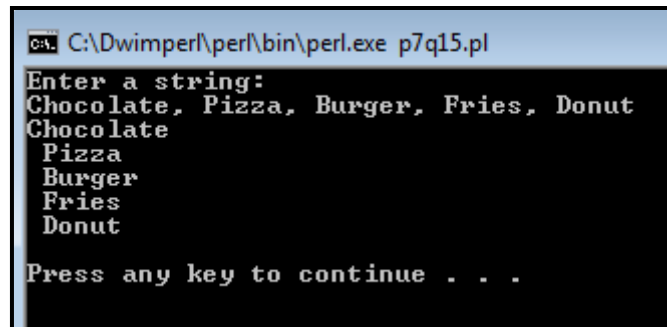
CODE:

```

print "Enter a string: \n";
$str = <stdin>;
@array = split ('.', $str);
foreach $n (@array)
{
    print "$n \n";
}

```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p7q15.pl
Enter a string:
Chocolate, Pizza, Burger, Fries, Donut
Chocolate
Pizza
Burger
Fries
Donut

Press any key to continue . . .
```

Fig14. Output for Perl script to demonstrate on splitting on string

Practical No. 8

Perl Formating

AIM:

To understand and write Perl programs for Perl formatting.

THEORY:

Perl uses a writing template called a 'format' to output reports. To use the format feature of Perl, you have to define a format first and then you can use that format to write formatted data.

Define a Format

Following is the syntax to define a Perl format –

```
format FormatName =
fieldline
value_one, value_two, value_three
fieldline
value_one, value_two
.
```

Here **FormatName** represents the name of the format. The **fieldline** is the specific way, the data should be formatted. The values lines represent the values that will be entered into the field line. You end the format with a single period.

Next **fieldline** can contain any text or fieldholders. The fieldholders hold space for data that will be placed there at a later date. A fieldholder has the format –

```
@<<<<
```

This fieldholder is left-justified, with a field space of 5. You must count the @ sign and the < signs to know the number of spaces in the field. Other field holders include –

```
@>>>> right-justified
```

```
@||| centered
```

```
@####.## numeric field holder
```

```
@* multiline field holder
```

An example format would be –

```
format EMPLOYEE =
```

```
=====
```

```
@<<<<<<<<<<<<<<<<<<<<<<<<< @<<
```

```
$name $age
```

```
@#####.##
```

```
$salary
```

```
=====
```

```
.
```

Using the Format

In order to invoke this format declaration, we would use the **write** keyword –

```
write EMPLOYEE;
```

The problem is that the format name is usually the name of an open file handle, and the write statement will send the output to this file handle. As we want the data sent to the STDOUT, we must associate EMPLOYEE with the STDOUT filehandle. First, however, we must make sure that that STDOUT is our selected file handle, using the select() function.

```
select(STDOUT);
```

We would then associate EMPLOYEE with STDOUT by setting the new format name with STDOUT, using the special variable \$~ or \$FORMAT_NAME as follows –

```
$~ = "EMPLOYEE";
```

When we now do a write(), the data would be sent to STDOUT. Remember: if you are going to write your report in any other file handle instead of STDOUT then you can use select() function to select that file handle and rest of the logic will remain the same.

Define a Report Header

Everything looks fine. But you would be interested in adding a header to your report. This header will be printed on top of each page. It is very simple to do this. Apart from defining a template you would have to define a header and assign it to \$^ or \$FORMAT_TOP_NAME variable

```
format EMPLOYEE_TOP =
```

```
=====
```

```
Name          Age
```

```
=====
```

```
.
```

Define a Report Footer

While \$^ or \$FORMAT_TOP_NAME contains the name of the current header format, there is no corresponding mechanism to automatically do the same thing for a footer. If you have a fixed-size footer, you can get footers by checking variable \$- or \$FORMAT_LINES_LEFT before each write() and print the footer yourself if necessary using another format defined as follows –

```
format EMPLOYEE_BOTTOM =
```

```
End of Page @<
```

```
    $%
```

```
.
```

Q.1) Write a Perl script to display text at center.

CODE:

```
$str = "Hello World";
```

```
format STDOUT =
```

```
@||||||||||||||||||||||||||||||||||||||||||
```

```
$str
```

```
.
```

```
write;
```

OUTPUT:

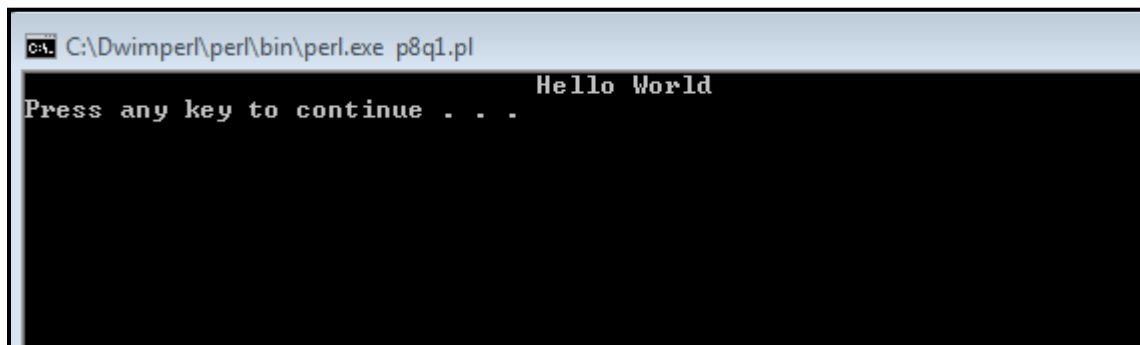


Fig1. Output for Perl script to display text at center

Q.2) Write a Perl script to accept a number and display using Perl formatter.

CODE:

```
print "Enter a number: \n";
```

```
$num = <stdin>;
```

```
format STDOUT =
```

```
=====
```

```
@#####.##
```

```
$num
```

```
=====
```

```
.
```

```
write;
```

OUTPUT:

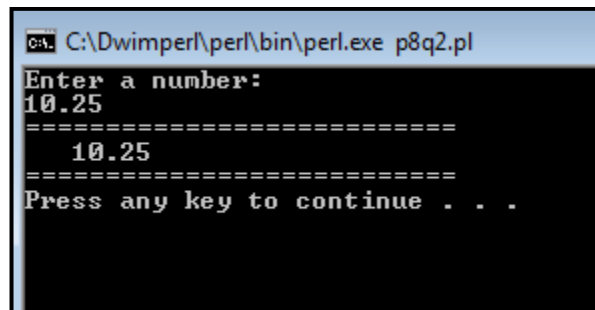


Fig2. Output for Perl script to accept a number and display using Perl formatter

Q.3) Write a Perl script to accept an array and display the values using Perl formatter.

CODE:

```
print "Enter 4 numbers: \n";
```

```
@num;
```

```
for ($i=0; $i<4; $i++)
```

```
{
```

```
    $num[$i] = <stdin>;
```

```
}
```

```
foreach $n(@num)
```

```
{
```

```
format STDOUT =
```

```
=====
```

```
@#####.##
```

\$n

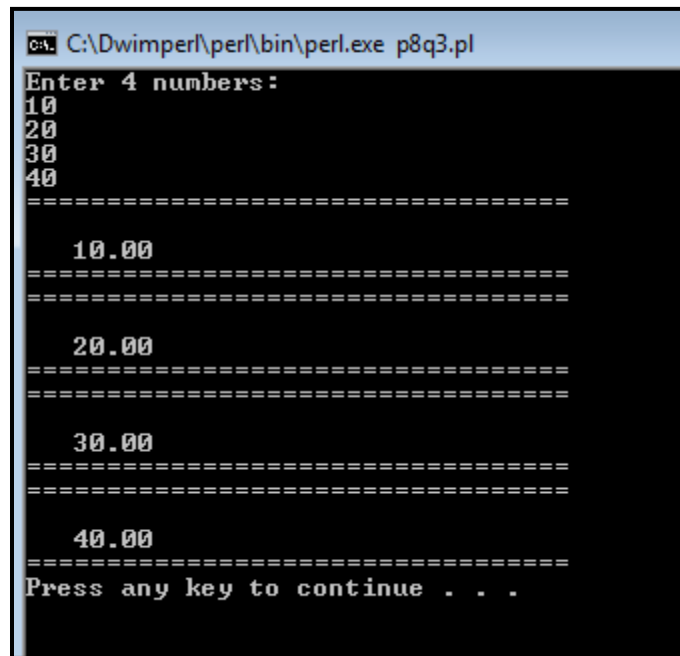
=====

.

write;

}

OUTPUT:



The screenshot shows a command prompt window titled "C:\Dwimper\perl\bin\perl.exe p8q3.pl". The prompt asks "Enter 4 numbers:". The user enters the numbers 10, 20, 30, and 40 on separate lines. The output displays each number followed by ".00", with each pair of lines separated by a line of equals signs. The output is: 10.00, 20.00, 30.00, 40.00. At the bottom, it says "Press any key to continue . . .".

Fig3. Output for Perl script to accept an array and display the values using Perl formatter

Q.4) Write a Perl script to accept a hash and display only keys using right/center formatting.

CODE:

```
%hash;
```

```
for($i=1; $i<4; $i++)
```

```
{
```

```
    print "Enter your name: \n";
```

```
    $name = <stdin>;
```

```
    print "Enter your rollno: \n";
```

```
    $rollno = <stdin>;
```

```
    $hash{$rollno} = $name;
```

```
}
```



```

@array = keys%hash;

foreach $n(@array)
{
format STDOUT =

=====

@|||||||

$n

=====

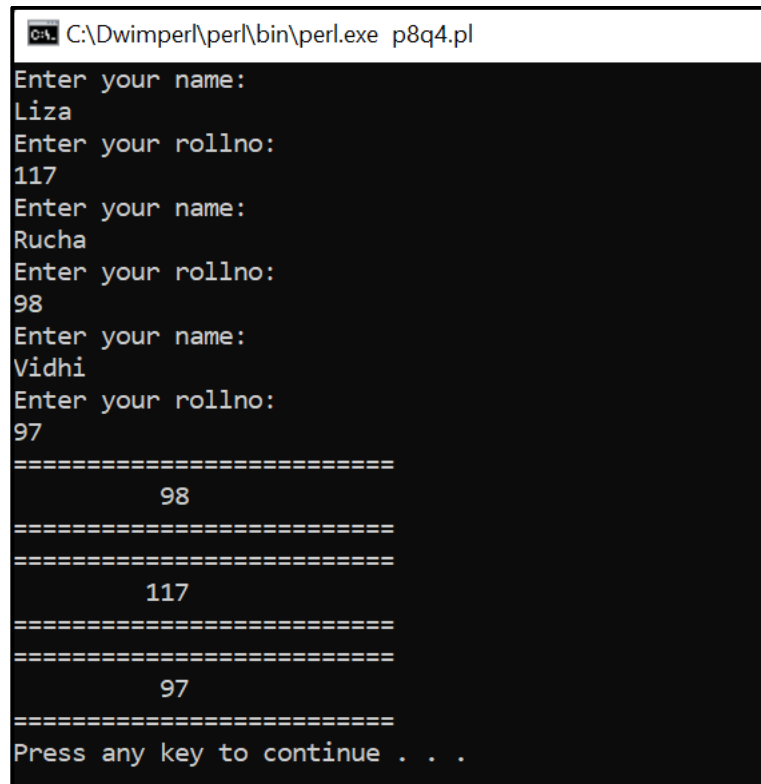
.

write;

}

```

OUTPUT:



```

C:\Dwimper\perl\bin\perl.exe p8q4.pl
Enter your name:
Liza
Enter your rollno:
117
Enter your name:
Rucha
Enter your rollno:
98
Enter your name:
Vidhi
Enter your rollno:
97
=====
          98
=====
          117
=====
          97
=====
Press any key to continue . . .

```

Fig4. Output for Perl script to accept a hash and display only keys using right/center formatting

Q.5) Write a Perl script to accept a decimal number and display only 4 values after decimal using formatting.

CODE:

```

print "Enter 4 numbers: \n";

@num;

for ($i=0; $i<4; $i++)
{
    $num[$i] = <stdin>;
}

foreach $n(@num)
{
    format STDOUT =

=====

@#####.#####

$n

=====

.

write;

}

```

OUTPUT:

```
C:\Dwimper\perl\bin\perl.exe p8q5.pl
Enter 4 numbers:
10.00
12.123
25
20.1
=====
10.0000
=====
12.1230
=====
25.0000
=====
20.1000
=====
Press any key to continue . . .
```

Fig5. Output for Perl script to accept a decimal number and display only 4 values after decimal using formatting.

Q.6) Write a Perl script to display name age and salary by name employee using formatting.

CODE:

format Employee =

=====

@<<<<<< @<<<<<<

\$name, \$age

@####.##

\$salary

=====

.

select (STDOUT);

\$~ = Employee;

@n = ("Liza", "Rucha", "Vidhi");

@a = ("23", "22", "21");

@s = ("5000", "5000", "5000");

for (\$i=0; \$i<3; \$i++)

```
{
$name=$n [$i];
$age=$a [$i];
$salary=$s [$i];

write;

}
```

OUTPUT:

```
C:\Dwimper\perl\bin\perl.exe p8q6.pl

=====
Liza    23
5000.00
=====
=====
Rucha   22
5000.00
=====
=====
Vidhi   21
5000.00
=====
Press any key to continue . . .
```

Fig6. Output for Perl script to display name age and salary by name employee using formatting

Q.7) Write a Perl script to display name, age and salary by name employee using REPORT formatting.

CODE:

```
format Employee =
=====

@<<<<<< @<<<<<< @#####.##

$name, $age, $salary

=====

.

format Employee_TOP =
=====

Name    Age    Salary

=====
```

```

select (STDOUT);

$~ = Employee;

$^ = Employee_TOP;

@n = ("Liza", "Rucha", "Vidhi");

@a = ("23", "22", "21");

@s = ("5000", "5000", "5000");

for ($i=0; $i<3; $i++)

{

$name=$n [$i];

$age=$a [$i];

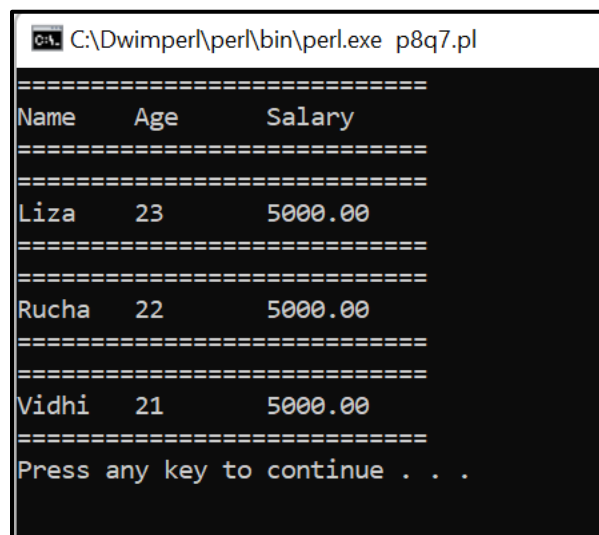
$salary=$s [$i];

write;

}

```

OUTPUT:



The screenshot shows a Windows command prompt window with the title bar "C:\Dwimper\perl\bin\perl.exe p8q7.pl". The output of the Perl script is displayed in a table format with columns for Name, Age, and Salary. The data is as follows:

Name	Age	Salary
Liza	23	5000.00
Rucha	22	5000.00
Vidhi	21	5000.00

Below the table, the prompt "Press any key to continue . . ." is visible.

Fig 7. Output for Perl script to display name, age and salary by name employee using REPORT formatting

Q.8) Write a Perl script to display patient_name, patient_age, health_issue and cost_of_treatment of a Patient_details using formatting.

CODE:

```
format Patient_details =
```

```
=====
```

```
@|||||||
```

```
$name
```

```
@<<<<<<<<, @<<<<<<<<, @####.##
```

```
$age, $issue, $cost
```

```
=====
```

```
.
```

```
select (STDOUT);
```

```
$~ = Patient_details;
```

```
@p_name = ("Liza", "Rucha", "Vidhi");
```

```
@p_age = ("23", "22", "21");
```

```
@p_health_issue = ("COVID", "Tyhpoid", "Cancer");
```

```
@p_cost = ("5000", "7000", "50000");
```

```
for ($i=0; $i<3; $i++)
```

```
{
```

```
$name=$p_name [$i];
```

```
$age=$p_age [$i];
```

```
$issue=$p_health_issue [$i];
```

```
$cost=$p_cost [$i];
```

```
write;
```

```
}
```

OUTPUT:

```
C:\Dwimper\perl\bin\perl.exe p8q8.pl

=====
                Liza
23            , COVID            ,      5000.00
=====
=====
                Rucha
22            , Tyhpoid          ,      7000.00
=====
=====
                Vidhi
21            , Cancer           ,     50000.00
=====
Press any key to continue . . .
```

Fig8. Output for Perl script to display patient_name, patient_age, health_issue and cost_of_treatment of a Patient_details using formatting

Q.9) Write a Perl script to display sequence_id, Sequence and alphabet_name using REPORT formatting.

CODE:

format Sequence =

[illegible]

\$id, \$seq, \$alphabet_name

•

format Sequence_TOP =

Sequence_id	Sequence	Alphabet_name
-------------	----------	---------------

•

```
select (STDOUT);
```

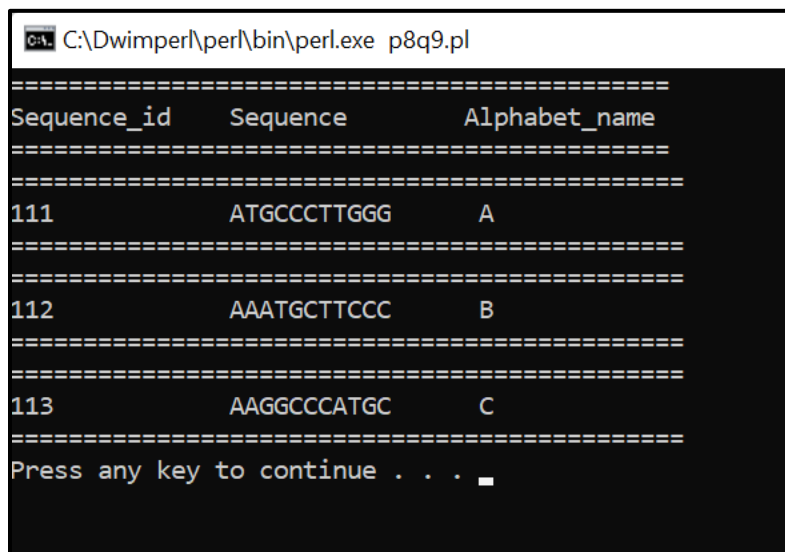
 $\$ \sim = \text{Sequence};$
$$\$^{\wedge} = \text{Sequence_TOP};$$

```

@i = ("111", "112", "113");
@s = ("ATGCCCTTGGG", "AAATGCTTCCC", "AAGGCCCATGC");
@a = ("A", "B", "C");
for ($i=0; $i<3; $i++)
{
    $id=$i [$i];
    $seq=$s [$i];
    $alphabet_name=$a [$i];
    write;
}

```

OUTPUT:



```

C:\Dwimperl\perl\bin\perl.exe p8q9.pl
=====
Sequence_id    Sequence      Alphabet_name
=====
111            ATGCCCTTGGG   A
=====
112            AAATGCTTCCC   B
=====
113            AAGGCCCATGC   C
=====
Press any key to continue . . .

```

Fig9. Output for Perl script to display sequence_id, Sequence and alphabet_name using REPORT formatting

Practical No. 9

OOPS IN PERL

AIM:

To understand the Object-Oriented Programming in Perl language.

THEORY:

Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Object Oriented concept in Perl is very much based on references and anonymous array and hashes.

There are three main terms, explained from the point of view of how Perl handles objects. The terms are object, class, and method.

- An **object** within Perl is merely a reference to a data type that knows what class it belongs to. The object is stored as a reference in a scalar variable. Because a scalar only contains a reference to the object, the same scalar can hold different objects in different classes.
- A **class** within Perl is a package that contains the corresponding methods required to create and manipulate objects.
- A **method** within Perl is a subroutine, defined with the package. The first argument to the method is an object reference or a package name, depending on whether the method affects the current object or the class.

Perl provides a `bless ()` function, which is used to return a reference which ultimately becomes an object. The [my](#) keyword is an access specifier which is localizing `$class` and `$self` to be within the enclosed block. [shift](#) keyword takes the package name from the default array “`@_`” and pass it on to the `bless` function.

The following codes will not Run-on Online IDE because of the use of Packages. The code below is a Perl class or a module file. Save the below file as (*.pm) extension.

Now, save the program that is used to get access to the attributes defined in the package by the name *.pl. Here, * can be any name (In this case it is test.pl).

Run the code saved as test.pl in the Perl command line by using the command `Perl test.pl`

Defining a Class

It is very simple to define a class in Perl. A class is corresponding to a Perl Package in its simplest form. To create a class in Perl, we first build a package.

A package is a self-contained unit of user-defined variables and subroutines, which can be re-used over and over again.

Perl Packages provide a separate namespace within a Perl program which keeps subroutines and variables independent from conflicting with those in other packages.

To declare a class named Person in Perl we do - **package Person;**

The scope of the package definition extends to the end of the file, or until another package keyword is encountered.

Creating and Using Objects

To create an instance of a class (an object) we need an object constructor. This constructor is a method defined within the package. Most programmers choose to name this object constructor method new, but in Perl you can use any name.

You can use any kind of Perl variable as an object in Perl. Most Perl programmers choose either references to arrays or hashes.

Defining Methods

Other object-oriented languages have the concept of security of data to prevent a programmer from changing an object data directly and they provide accessor methods to modify object data. Perl does not have private variables but we can still use the concept of helper methods to manipulate object data.

Q.1) Write a Perl script to create class Square and find area of square.

CODE:

```
package square;
```

```
sub new
```

```
{  
    $class = shift;  
    $ref = {side=>shift};  
    bless $ref,$class;  
    return $ref;  
}
```

```
sub area
```

```
{  
    $ref = shift;  
    return $ref->{side}*$ref->{side};  
}
```

```

1;
use square;
$obj = new square(10);
print "Area of square is ", $obj->area();
print "\n";

```

OUTPUT:

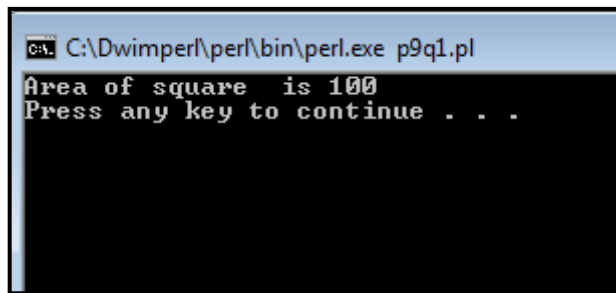


Fig1. Output for Perl script to create class Square and find area of square

Q.2 Write a Perl script to Create a student class and initialize it with name and roll number by using hash reference. Make methods to Display – It should display all information's of the student.

CODE:

```

package student;

sub new
{
    $class = shift;

    $ref = { name=>shift, rollnumber=>shift };

    bless $ref,$class;

    return $ref;
}

sub display
{
    $ref = shift;

    print "Student name is ", $ref->{ name }, " and roll number is ",$ref->{rollnumber};

    print "\n";
}

```

```
1;  
use student;  
$obj = new student(Liza,117);  
print $obj->display();
```

OUTPUT:

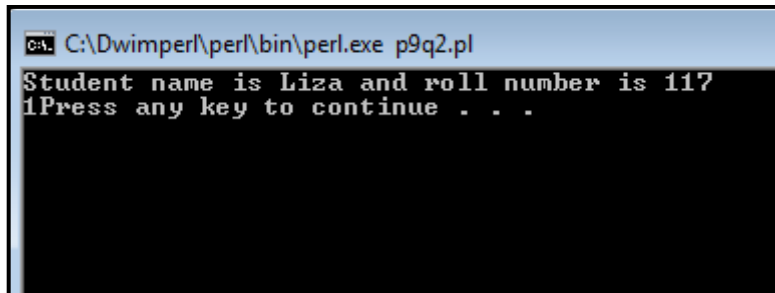
A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Dwimper\perl\bin\perl.exe p9q2.pl'. The command prompt displays the output of a Perl script: 'Student name is Liza and roll number is 117' followed by 'Press any key to continue . . .' on the next line. The background is black, and the text is white.

Fig2. Output for a Perl script to Create a student class and initialize it with name and roll number by using hash reference and display all information's of the student

Q.3) Write a Perl script to create class Perimeter to calculate perimeter of a square and triangle using inheritance.

CODE:

```
package Square_peri;  
  
sub new{  
    $class = shift;  
    $ref = { };  
  
    bless $ref, $class;  
    return $ref;  
}  
  
sub sq_peri{  
    $ans, $side;  
    $side=26;  
    $ans = 4*$side;
```

```

        print("Perimeter of the square is $ans\n");
    }
1;

package Triangle_peri;

sub new{
    $class = shift;
    $ref = { };

    bless $ref, $class;
    return $ref;
}

sub tri_peri{
    $ans, $side1, $side2, $side3;
    $side1=24;
    $side2=53;
    $side3=24;
    $ans = $side1+$side2+$side3;
    print("Perimeter of the triangle is $ans\n");
}
1;

package Perimeter;

use parent 'Square_peri';
use parent 'Triangle_peri';

sub new{

```

```

$class = shift;

$ref = {};

bless $ref, $class;

return $ref;
}

1;

use Perimeter;

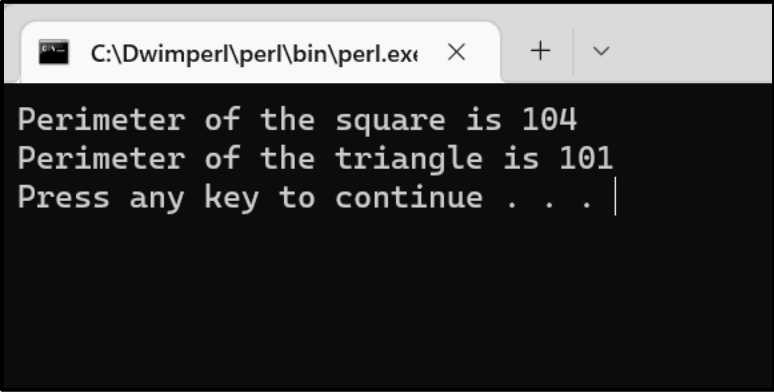
$obj = new Perimeter();

$obj->sq_peri;

$obj->tri_peri;

```

OUTPUT:



The screenshot shows a Windows command prompt window with the title bar 'C:\Dwimperl\perl\bin\perl.exe'. The output text is as follows:

```

Perimeter of the square is 104
Perimeter of the triangle is 101
Press any key to continue . . . |

```

Fig3. Output for a Perl script to create class Perimeter to calculate perimeter of a square and triangle using inheritance

Q.4) Write a Perl script to Print the average of three numbers entered by user by creating a class named “Average” having a method to calculate and print the average.

CODE:

```

package Average;

sub new{

    $class = shift;

    $ref = {a => shift, b => shift, c => shift};

```

```

        bless $ref, $class;

        return $ref;
    }

sub calculate{

    $ref = shift;

    $a = $ref -> {a};

    $b = $ref -> {b};

    $c = $ref -> {c};

    $avg = $a + $b + $c / 3;

    print("The everage of the entered numbers is $avg\n");

}

1;

use Average;

print("Enter #1: ");

$a = <stdin>;

print("Enter #2: ");

$b = <stdin>;

print("Enter #3: ");

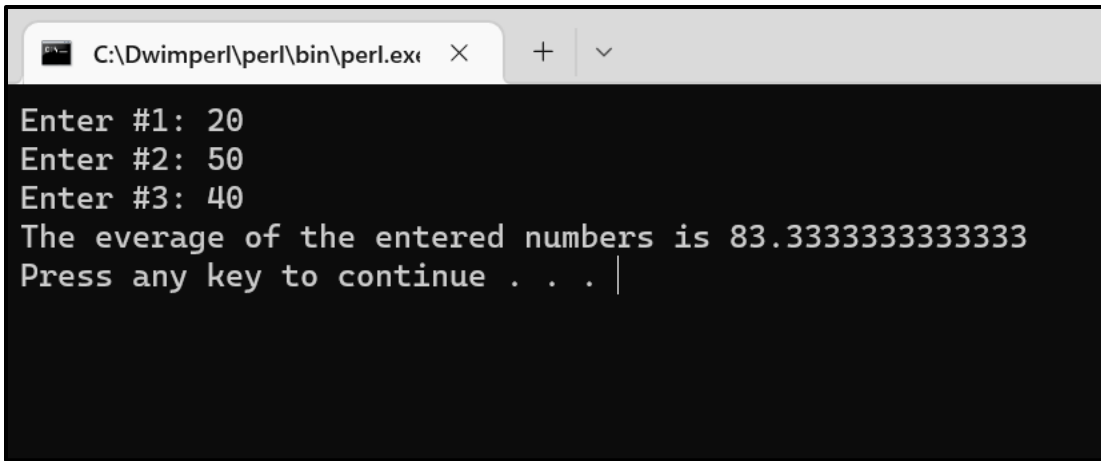
$c = <stdin>;

$obj = new Average($a, $b, $c);

$obj -> calculate();

```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Dwimperl\perl\bin\perl.exe' and standard window controls. The command prompt displays the following text: 'Enter #1: 20', 'Enter #2: 50', 'Enter #3: 40', 'The average of the entered numbers is 83.3333333333333', and 'Press any key to continue . . . |'.

```
C:\Dwimperl\perl\bin\perl.exe X + v
Enter #1: 20
Enter #2: 50
Enter #3: 40
The average of the entered numbers is 83.3333333333333
Press any key to continue . . . |
```

Fig4. Output for a Perl script to Print the average of three numbers entered by user by creating a class named “Average” having a method to calculate and print the average

Q.5) Write a program that would print the information (name, year of joining, salary, address) of three employees by creating a class named “Employee”

The output should be as follows:

Name	Year of joining	Address
Robert	1994	64C- WallsStreat
Sam	2000	68D- WallsStreat
John	1999	26B- WallsStreat

CODE:

```
package Employee;

sub new
{
    $class = shift;

    $ref = { name=>shift, year=>shift, address=>shift };

    bless $ref,$class;

    return $ref;
}

sub display
```



```

{
    $ref = shift;
    print $ref->{name}, "\t\t";
    print $ref->{year}, "\t\t";
    print $ref->{address}, "\t\t";
    print "\n";
}

1;

use Employee;

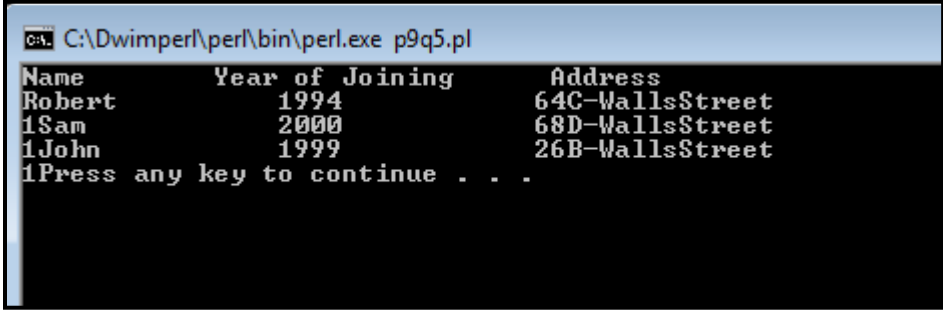
print "Name      Year of Joining   Address \n";

$obj1 = new Employee("Robert", 1994, "64C-WallsStreet");
$obj2 = new Employee("Sam", 2000, "68D-WallsStreet");
$obj3 = new Employee("John", 1999, "26B-WallsStreet");

print $obj1->display();
print $obj2->display();
print $obj3->display();

```

OUTPUT:



```

C:\Dwimper\perl\bin\perl.exe p9q5.pl
Name      Year of Joining   Address
Robert    1994              64C-WallsStreet
1Sam      2000              68D-WallsStreet
1John     1999              26B-WallsStreet
1Press any key to continue . . .

```

Fig5. Output for a program that would print the information (name, year of joining, salary, address) of three employees by creating a class named “Employee”

Q.6) Write a Perl script create class and display only DNA sequence by applying conditions.

CODE:

```

package Dna;

```

```

sub new{

    $class = shift;

    $ref = { seq => shift };

    bless $ref, $class;

    return $ref;

}

sub display{

    $ref = shift;

    $seq = $ref -> {seq};

    if($seq =~ m/[^atgc]/g){

        print "DNA sequence is not present\n";

    }else{

        print "DNA sequence present is $seq\n";

    }

}

1;

use Dna;

$dna_seq = "atgccgttttggcgaggtaataaa";

$rna_seq = "auuugggccguguauuaauggcuc";

$obj = new Dna($dna_seq);

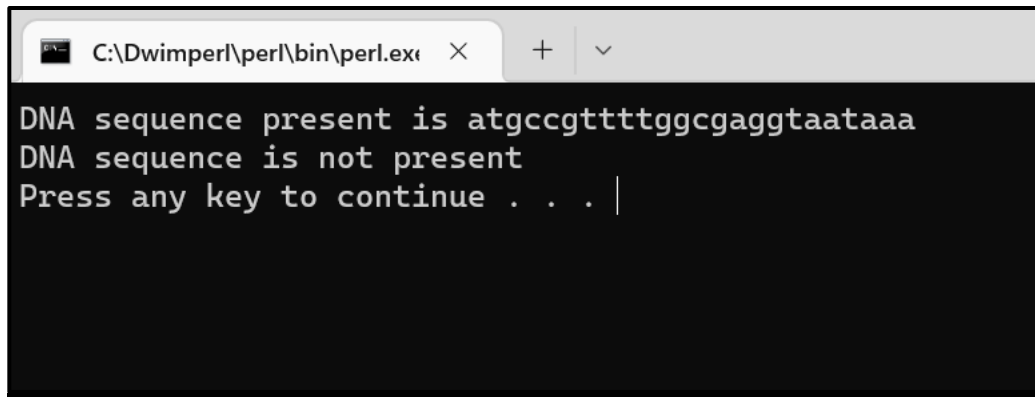
$obj -> display();

$obj = new Dna($rna_seq);

$obj -> display();

```

OUTPUT:

A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Dwimperl\perl\bin\perl.exe' and standard window controls. The command prompt displays three lines of text: 'DNA sequence present is atgccgttttggcgaggtaataaa', 'DNA sequence is not present', and 'Press any key to continue . . . |'. The cursor is positioned at the end of the third line.

```
C:\Dwimperl\perl\bin\perl.exe X + v
DNA sequence present is atgccgttttggcgaggtaataaa
DNA sequence is not present
Press any key to continue . . . |
```

Fig6. Output for a Perl script create class and display only DNA sequence by applying conditions

Practical No. 10

DBI IN PERL

AIM:

To understand the Database Independent Interface for Perl language.

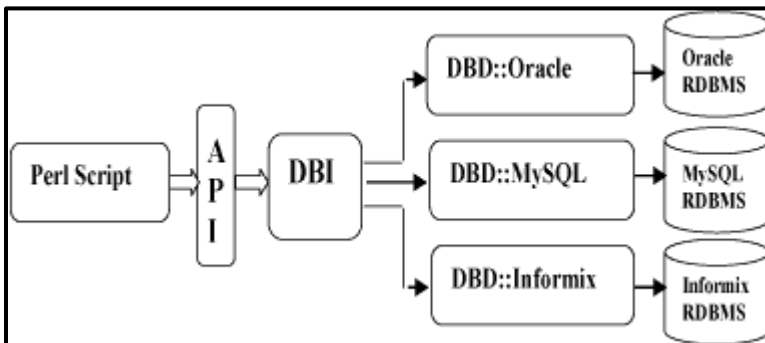
THEORY:

DBI stands for Database Independent Interface for Perl, which means DBI provides an abstraction layer between the Perl code and the underlying database, allowing you to switch database implementations really easily.

The DBI is a database access module for the Perl programming language. It provides a set of methods, variables, and conventions that provide a consistent database interface, independent of the actual database being used.

Architecture of a DBI Application

DBI is independent of any database available in backend. You can use DBI whether you are working with Oracle, MySQL or Informix, etc. This is clear from the following architecture diagram.



Here DBI is responsible of taking all SQL commands through the API, (i.e., Application Programming Interface) and to dispatch them to the appropriate driver for actual execution. And finally, DBI is responsible of taking results from the driver and giving back it to the calling script.

Notation and Conventions

Throughout this chapter following notations will be used and it is recommended that you should also follow the same convention.

- \$dsn Database source name
- \$dbh Database handle object
- \$sth Statement handle object
- \$h Any of the handle types above (\$dbh, \$sth, or \$drh)
- \$rc General Return Code (boolean: true=ok, false=error)
- \$rv General Return Value (typically an integer)
- @ary List of values returned from the database.
- \$rows Number of rows processed (if available, else -1)

- `$fh` A filehandle
- `undef` NULL values are represented by undefined values in Perl
- `\%attr` Reference to a hash of attribute values passed to methods

Database Connection

Assuming we are going to work with MySQL database. Before connecting to a database make sure of the followings. You can take help of our MySQL tutorial in case you are not aware about how to create database and tables in MySQL database.

- You have created a database with a name TESTDB.
- You have created a table with a name TEST_TABLE in TESTDB.
- This table is having fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.
- User ID "testuser" and password "test123" are set to access TESTDB.
- Perl Module DBI is installed properly on your machine.
- You have gone through MySQL tutorial to understand MySQL Basics.

INSERT Operation

INSERT operation is required when you want to create some records into a table. Here we are using table TEST_TABLE to create our records. So once our database connection is established, we are ready to create records into TEST_TABLE. Following is the procedure to create single record into TEST_TABLE. You can create as many as records you like using the same concept.

Record creation takes the following steps –

- Preparing SQL statement with INSERT statement. This will be done using **prepare()** API.
- Executing SQL query to select all the results from the database. This will be done using **execute()** API.
- Releasing Statement handle. This will be done using **finish()** API.
- If everything goes fine then **commit** this operation otherwise you can **rollback** complete transaction. Commit and Rollback are explained in next sections.

Using Bind Values

There may be a case when values to be entered is not given in advance. So you can use bind variables which will take the required values at run time. Perl DBI modules make use of a question mark in place of actual value and then actual values are passed through execute() API at the run time.

READ Operation

READ Operation on any database means to fetch some useful information from the database, i.e., one or more records from one or more tables. So once our database connection is established, we are ready to make a query into this database. Following is the procedure to query all the records having AGE greater than 20.

This will take four steps –

- Preparing SQL SELECT query based on required conditions. This will be done using **prepare()** API.

- Executing SQL query to select all the results from the database. This will be done using **execute()** API.
- Fetching all the results one by one and printing those results. This will be done using **fetchrow_array()** API.
- Releasing Statement handle. This will be done using **finish()** API.

UPDATE Operation

UPDATE Operation on any database means to update one or more records already available in the database tables. Following is the procedure to update all the records having SEX as 'M'. Here we will increase AGE of all the males by one year. This will take three steps –

- Preparing SQL query based on required conditions. This will be done using **prepare()** API.
- Executing SQL query to select all the results from the database. This will be done using **execute()** API.
- Releasing Statement handle. This will be done using **finish()** API.
- If everything goes fine then **commit** this operation otherwise you can **rollback** complete transaction. See next section for commit and rollback APIs.

DELETE Operation

DELETE operation is required when you want to delete some records from your database. Following is the procedure to delete all the records from TEST_TABLE where AGE is equal to 30. This operation will take the following steps.

- Preparing SQL query based on required conditions. This will be done using **prepare()** API.
- Executing SQL query to delete required records from the database. This will be done using **execute()** API.
- Releasing Statement handle. This will be done using **finish()** API.
- If everything goes fine then **commit** this operation otherwise you can **rollback** complete transaction.

Q.1) Write a Perl script to make a connection with MySQL for any database.

CODE:

```
use DBI;
```

```
$dbh = DBI->connect("DBI:mysql:Company", "root", "1234") or die $DBI::errstr;
```

```
print("Database is connected succesfully \n");
```

OUTPUT:

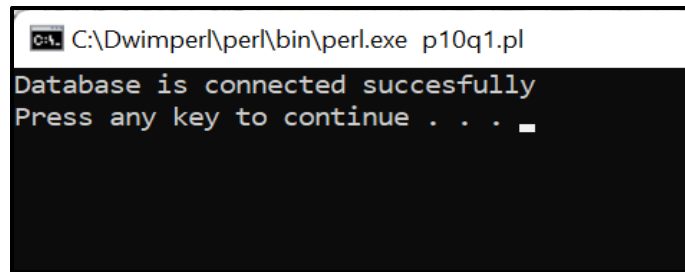


Fig1. Output for Perl script to make a connection with MySQL for any database

Q.2) Write a Perl Script to create table Employee under database Company.

CODE:

```
use DBI;

$dbh = DBI->connect("DBI:mysql:Company", "root", "1234") or die $DBI::errstr;

print("Database is connected succesfully \n");

$sth = $d

bh->prepare("CREATE TABLE Employee(ID int, NAME varchar(20), Salary int)");

$sth->execute() or die $DBI::errstr;

$sth->finish();

print "Table Employee created under Company Database \n";
```

OUTPUT:

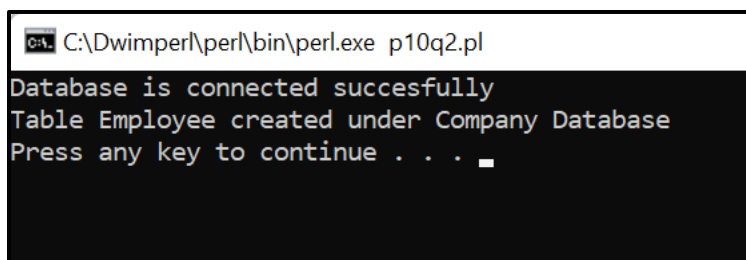


Fig2. Perl Script to create table Employee under database Company

Q.3) Write a Perl Script to insert two values (one without binding Values and one with binding Values respectively).

CODE:

```
use DBI;

$dbbh = DBI->connect("DBI:mysql:Company", "root", "1234") or die $DBI::errstr;

print "Database is connected successfully\n";
```

```

$sth = $dbh->prepare("insert into Employee(ID, NAME, SALARY) values (1, 'Liza Patel', 50000)");
$sth->execute() or die DBI::errstr;
$sth->finish();

print "Values inserted into Table without using Binding values\n";

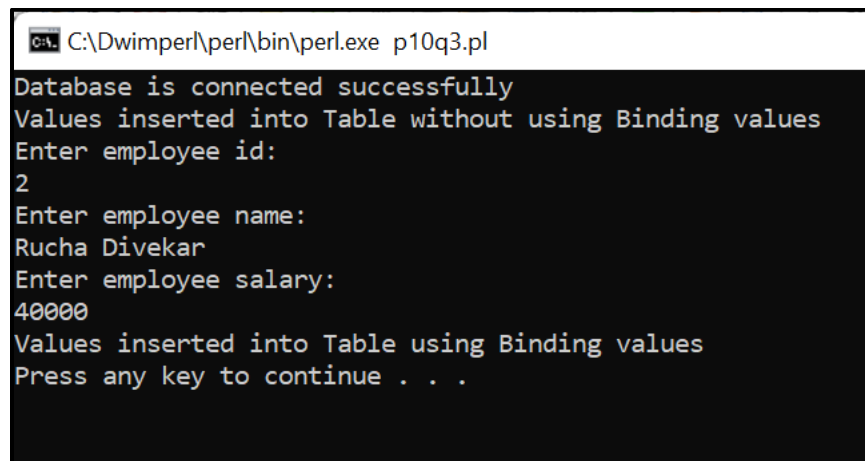
print "Enter employee id: \n";
$id=<stdin>;
print "Enter employee name: \n";
$name=<stdin>;
print "Enter employee salary: \n";
$salary=<stdin>;

$sth = $dbh->prepare("insert into Employee(ID, NAME, SALARY) values (?, ?, ?)");
$sth->execute($id, $name, $salary);

print("Values inserted into Table using Binding values\n");

```

OUTPUT:



```

C:\Dwimper\perl\bin\perl.exe p10q3.pl
Database is connected successfully
Values inserted into Table without using Binding values
Enter employee id:
2
Enter employee name:
Rucha Divekar
Enter employee salary:
40000
Values inserted into Table using Binding values
Press any key to continue . . .

```

Fig3.1 Output for Perl Script to insert two values (one without binding Values and one with binding Values respectively).


```
mysql> use Company
Database changed
mysql> select * from Employee;
+-----+-----+-----+
| ID   | NAME      | Salary |
+-----+-----+-----+
| 1    | Liza Patel | 50000  |
| 2    | Rucha Divekar | 40000  |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

Fig3.2 Output in MySQL

Q.4) Write a Perl Script to display all data from employee table.

CODE:

```
use DBI;
```

```
$dbh = DBI->connect("DBI:mysql:Company", "root", "1234") or die $DBI::errstr;
```

```
print("Database connected\n");
```

```
$sth = $dbh->prepare("select * from Employee");
```

```
$sth->execute();
```

```
while(my @row = $sth->fetchrow()){
```

```
    my($id, $name, $salary) = @row;
```

```
    print("
```

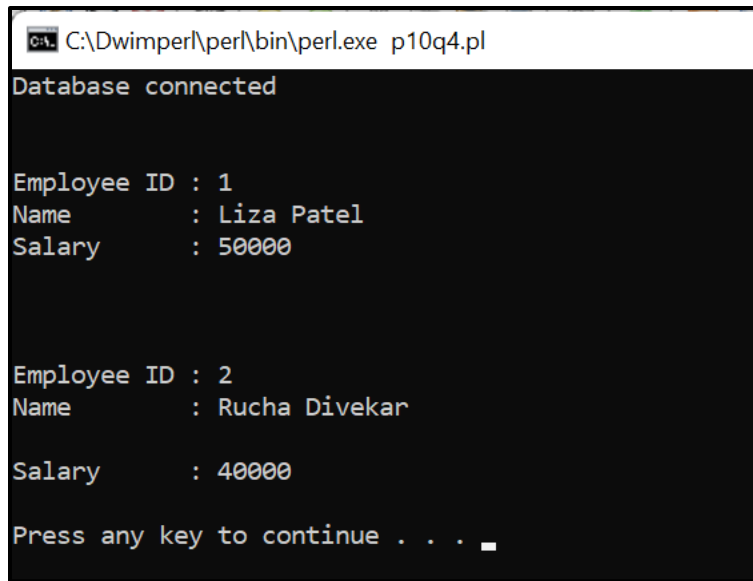
```
Employee ID : $id
```

```
Name      : $name
```

```
Salary    : $salary
```

```
\n");} $sth->finish();
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p10q4.pl
Database connected

Employee ID : 1
Name       : Liza Patel
Salary     : 50000

Employee ID : 2
Name       : Rucha Divekar
Salary     : 40000

Press any key to continue . . . _
```

Fig4. Output for Perl Script to display all data from employee table

Q 5 Write a Perl Script to display the employee data who salary greater than entered salary using binding values (ask user to enter).

CODE:

use DBI;

\$dbh = DBI->connect("DBI:mysql:Company", "root", "1234") or die \$DBI::errstr;

print("Database connected\n");

print "Enter salary: \n";

\$salary = <stdin>;

\$sth = \$dbh->prepare("select * from Employee where Salary>?");

\$sth->execute(\$salary) or die \$DBI::errstr;

while(my @row = \$sth->fetchrow()){

 my(\$id, \$name, \$salary) = @row;

 print("

Employee ID : \$id

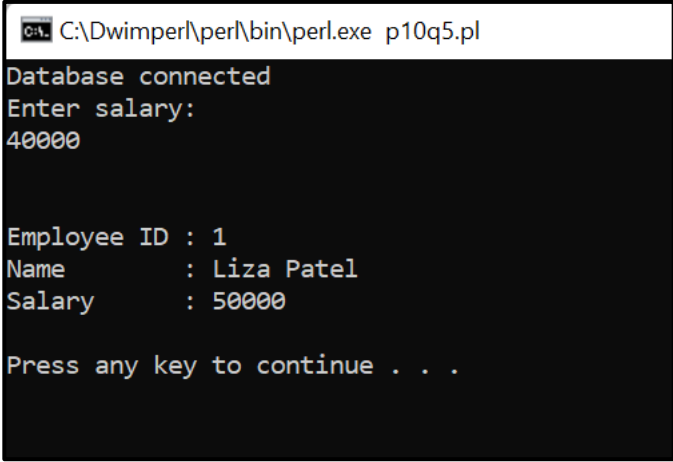
Name : \$name

Salary : \$salary

\n");

} \$sth->finish();

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p10q5.pl
Database connected
Enter salary:
40000

Employee ID : 1
Name      : Liza Patel
Salary    : 50000

Press any key to continue . . .
```

Fig5. Output for a Perl Script to display the employee data who salary greater than entered salary using binding values (ask user to enter)

Q.6) Write a Perl Script to update a record of employee whose employee id is 4 and 5 using one without binding Values and one with binding Values respectively.

CODE:

```
use DBI;
```

```
$dbh = DBI->connect("DBI:mysql:Company", "root", "1234") or die $DBI::errstr;
```

```
print "Database is connected successfully\n";
```

```
$sth = $dbh->prepare("UPDATE Employee SET Salary = Salary + 2000 WHERE Id = 4");
```

```
$sth->execute() or die $DBI::errstr;
```

```
$sth->finish();
```

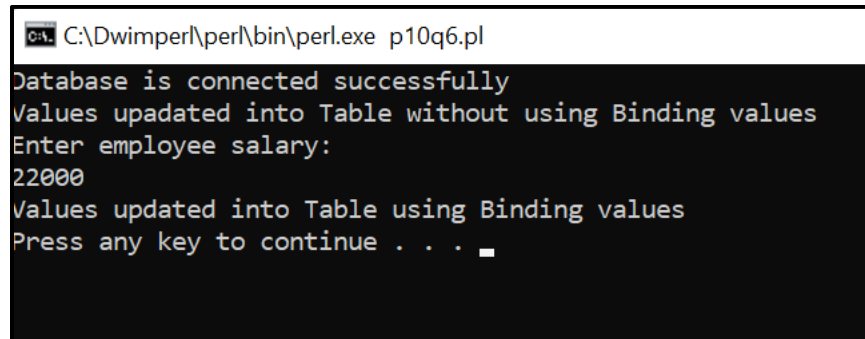
```
print "Values upadated into Table without using Binding values\n";
```

```
print "Enter employee salary: \n";
```

```
$salary=<stdin>;
```

```
$sth = $dbh->prepare("UPDATE Employee SET Salary = ? WHERE Id = 5");
$sth->execute($salary) or die DBI::errstr;
print "Values updated into Table using Binding values\n";
$sth->finish()
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p10q6.pl
Database is connected successfully
Values upadated into Table without using Binding values
Enter employee salary:
22000
Values updated into Table using Binding values
Press any key to continue . . .
```

Fig6.1. Output for a Perl Script to update a record of employee whose employee id is 4 and 5 using one without binding Values and one with binding Values respectively

```
mysql> select * from Employee;
```

ID	NAME	Salary
1	Liza Patel	50000
2	Rucha Divekar	40000
3	Vidhi Shah	30000
4	Shalmon Anandas	64000
5	Kaustubh Paturkar	22000

```
5 rows in set (0.00 sec)
```

Fig6.2 Output in MySQL

Q.7) Write a Perl Script to delete a record of employee whose employee id is 4 and 5 using one without binding Values and one with binding Values respectively.

CODE:

```
use DBI;

$dbh = DBI->connect("DBI:mysql:Company", "root", "1234") or die $DBI::errstr;
print "Database is connected successfully\n";

$sth = $dbh->prepare("DELETE from Employee WHERE Id = 4");
$sth->execute() or die DBI::errstr;
```

```

$sth->finish();

print "Values deleted from Table without using Binding values\n";

print "Enter employee id: \n";

$id=<stdin>;

$sth = $dbh->prepare("DELETE from Employee WHERE Id = ?");

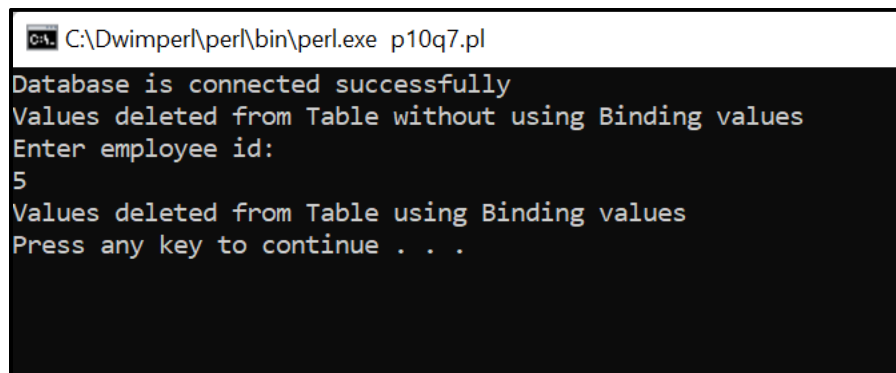
$sth->execute($id) or die DBI::errstr;

print "Values deleted from Table using Binding values\n";

$sth->finish();

```

OUTPUT:



```

C:\Dwimper\perl\bin\perl.exe p10q7.pl
Database is connected successfully
Values deleted from Table without using Binding values
Enter employee id:
5
Values deleted from Table using Binding values
Press any key to continue . . .

```

Fig7.1. Output for a Perl Script to delete a record of employee whose employee id is 4 and 5 using one without binding Values and one with binding Values respectively



```

mysql> select * from Employee;
+-----+-----+-----+
| ID | NAME          | Salary |
+-----+-----+-----+
| 1 | Liza Patel    | 50000 |
| 2 | Rucha Divekar | 40000 |
| 3 | Vidhi Shah    | 30000 |
+-----+-----+-----+
3 rows in set (0.00 sec)

```

Fig7.2. Output for MySQL

Q.8) Write a Perl Script to drop employee table.

CODE:

```

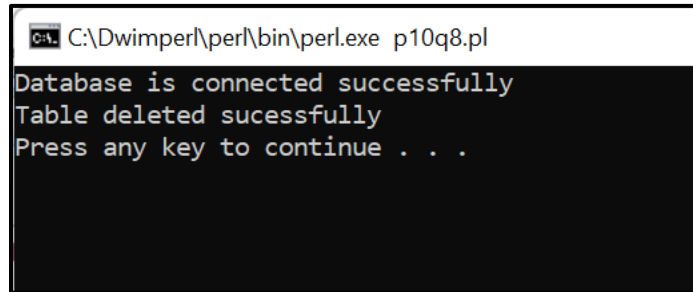
use DBI;

$dbh = DBI->connect("DBI:mysql:Company", "root", "1234") or die $DBI::errstr;

```

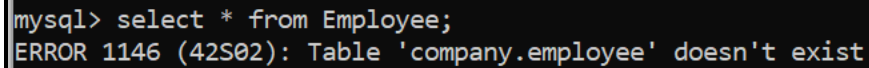
```
print "Database is connected successfully\n";  
$sth = $dbh->prepare("DROP table Employee");  
$sth->execute() or die DBI::errstr;  
print "Table deleted sucessfully \n";  
$sth->finish();
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p10q8.pl  
Database is connected successfully  
Table deleted sucessfully  
Press any key to continue . . .
```

Fig8.1. Output for a Perl Script to drop employee table



```
mysql> select * from Employee;  
ERROR 1146 (42S02): Table 'company.employee' doesn't exist
```

Fig8.2. Output for MySQL

Practical No. 11

File handling and Directory

AIM:

To understand and write perl programs for file handling and directory.

THEORY:

The basics of handling files are simple: you associate a filehandle with an external entity (usually a file) and then use a variety of operators and functions within Perl to read and update the data stored within the data stream associated with the filehandle.

A filehandle is a named internal Perl structure that associates a physical file with a name. All filehandles are capable of read/write access, so you can read from and update any file or device associated with a filehandle. However, when you associate a filehandle, you can specify the mode in which the filehandle is opened.

Opening and Closing Files

There are following two functions with multiple forms, which can be used to open any new or existing file in Perl.

```
open FILEHANDLE, EXPR
```

```
open FILEHANDLE
```

Here FILEHANDLE is the file handle returned by the open function and EXPR is the expression having file name and mode of opening the file.

Open Function

Following is the syntax to open file.txt in read-only mode. Here less than < sign indicates that file has to be opened in read-only mode.

```
open(DATA, "<file.txt");
```

Here DATA is the file handle, which will be used to read the file. Here is the example, which will open a file and will print its content over the screen.

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

```
open(DATA, "<file.txt") or die "Couldn't open file file.txt, $!";
```

```
while(<DATA>) {
```

```
    print "$_";
```

```
}
```

Following is the syntax to open file.txt in writing mode. Here less than > sign indicates that file has to be opened in the writing mode.

```
open(DATA, ">file.txt") or die "Couldn't open file file.txt, $!";
```

This example actually truncates (empties) the file before opening it for writing, which may not be the desired effect. If you want to open a file for reading and writing, you can put a plus sign before the > or < characters.

For example, to open a file for updating without truncating it –

```
open(DATA, "+<file.txt"); or die "Couldn't open file file.txt, $!";
```

To truncate the file first –

```
open DATA, "+>file.txt" or die "Couldn't open file file.txt, $!";
```

You can open a file in the append mode. In this mode, writing point will be set to the end of the file.

```
open(DATA, ">>file.txt") || die "Couldn't open file file.txt, $!";
```

A double >> opens the file for appending, placing the file pointer at the end, so that you can immediately start appending information. However, you can't read from it unless you also place a plus sign in front of it –

```
open(DATA, "+>>file.txt") || die "Couldn't open file file.txt, $!";
```

Following is the table, which gives the possible values of different modes

Sr.No.	Entities & Definition
1	< or r Read Only Access
2	> or w Creates, Writes, and Truncates
3	>> or a Writes, Appends, and Creates
4	+< or r+ Reads and Writes
5	+> or w+ Reads, Writes, Creates, and Truncates

6

+>> or a+

Reads, Writes, Appends, and Creates

Close Function

To close a filehandle, and therefore disassociate the filehandle from the corresponding file, you use the close function. This flushes the filehandle's buffers and closes the system's file descriptor.

close FILEHANDLE

close

If no FILEHANDLE is specified, then it closes the currently selected filehandle. It returns true only if it could successfully flush the buffers and close the file.

close(DATA) || die "Couldn't close file properly";

Reading and Writing Files

Once you have an open filehandle, you need to be able to read and write information. There are a number of different ways of reading and writing data into the file.

The <FILEHANDL> Operator

The main method of reading the information from an open filehandle is the <FILEHANDLE> operator. In a scalar context, it returns a single line from the filehandle. For example –

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

```
print "What is your name?\n";
```

```
$name = <STDIN>;
```

```
print "Hello $name\n";
```

When you use the <FILEHANDLE> operator in a list context, it returns a list of lines from the specified filehandle. For example, to import all the lines from a file into an array –

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

```
open(DATA,"<import.txt") or die "Can't open data";
```

```
@lines = <DATA>;
```

```
close(DATA);
```

getc Function

The getc function returns a single character from the specified FILEHANDLE, or STDIN if none is specified –

getc FILEHANDLE

getc

If there was an error, or the filehandle is at end of file, then undef is returned instead.

read Function

The read function reads a block of information from the buffered filehandle: This function is used to read binary data from the file.

read FILEHANDLE, SCALAR, LENGTH, OFFSET

read FILEHANDLE, SCALAR, LENGTH

The length of the data read is defined by LENGTH, and the data is placed at the start of SCALAR if no OFFSET is specified. Otherwise data is placed after OFFSET bytes in SCALAR. The function returns the number of bytes read on success, zero at end of file, or undef if there was an error.

Perl - Directories

Following are the standard functions used to play with directories.

opendir DIRHANDLE, EXPR # To open a directory

readdir DIRHANDLE # To read a directory

rewinddir DIRHANDLE # Positioning pointer to the beginning

telldir DIRHANDLE # Returns current position of the dir

seekdir DIRHANDLE, POS # Pointing pointer to POS inside dir

closedir DIRHANDLE # Closing a directory.

Display all the Files

There are various ways to list down all the files available in a particular directory. First let's use the simple way to get and list down all the files using the **glob** operator –

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

```
# Display all the files in /tmp directory.
```

```
$dir = "/tmp/*";
```

```
my @files = glob( $dir );
```

```
foreach ( @files ) {
```

```
    print $_ . "\n";
```

```
}
```

Create new Directory

You can use **mkdir** function to create a new directory. You will need to have the required permission to create a directory.

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

```
$dir = "/tmp/perl";
```

```
# This creates perl directory in /tmp directory.
```

```
mkdir( $dir ) or die "Couldn't create $dir directory, $!";
```

```
print "Directory created successfully\n";
```

Remove a directory

You can use **rmdir** function to remove a directory. You will need to have the required permission to remove a directory. Additionally this directory should be empty before you try to remove it.

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

```
$dir = "/tmp/perl";
```

```
# This removes perl directory from /tmp directory.
```

```
rmdir( $dir ) or die "Couldn't remove $dir directory, $!";
```

```
print "Directory removed successfully\n";
```

Change a Directory

You can use **chdir** function to change a directory and go to a new location. You will need to have the required permission to change a directory and go inside the new directory.

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

```
$dir = "/home";
```

```
# This changes perl directory and moves you inside /home directory.
```

```
chdir( $dir ) or die "Couldn't go inside $dir directory, $!";
```

```
print "Your new location is $dir\n";
```

Q.1) Write a Perl script to read the file named as dna.txt and display the sequence as an output without using `getc ()` and `read ()` function.

CODE:

```
$file = "C:/Users/lizap/Documents/dna.txt";
```

```
open(FH, '<', $file) or die "File can't be opened";
```

```
@seq = <FH>;
```

```
print("@seq");
```

```
close FH;
```

OUTPUT:

```
C:\Dwimper\perl\bin\perl.exe p11q1.pl
GCATTCTGAGGCATTCTCTAACAGGTTCTCGACCCTCCGCCATGGCCCCGTGGATGCATCTCCTCACCGT
GCTGGCCCTGCTGGCCCTCTGGGGACCCAACCTCTGTTTCAGGCCTATTCCAGCCAGCACCTGTGCGGCTCC
AACCTAGTGGAGGCACTGTACATGACATGTGGACGGAGTGGCTTCTATAGACCCACGACCGCCGAGAGC
TGGAGGACCTCCAGGTGGAGCAGGCAGAACTGGGTCTGGAGGCAGGCGGCCTGCAGCCTTCGGCCCTGGA
GATGATTCTGCAGAAGCGCGGCATTGTGGATCAGTGCTGTAATAACATTTGCACATTTAACCAGCTGCAG
AACTACTGCAATGTCCCTTAGACACCTGCCTTGGGCCTGGCCTGCTGCTCTGCCCTGGCAACCAATAAAC
CCCTTGAATGAG
Press any key to continue . . .
```

Fig1. Output for a Perl script to read the file named as dna.txt and display the sequence as an output without using `getc ()` and `read ()` function

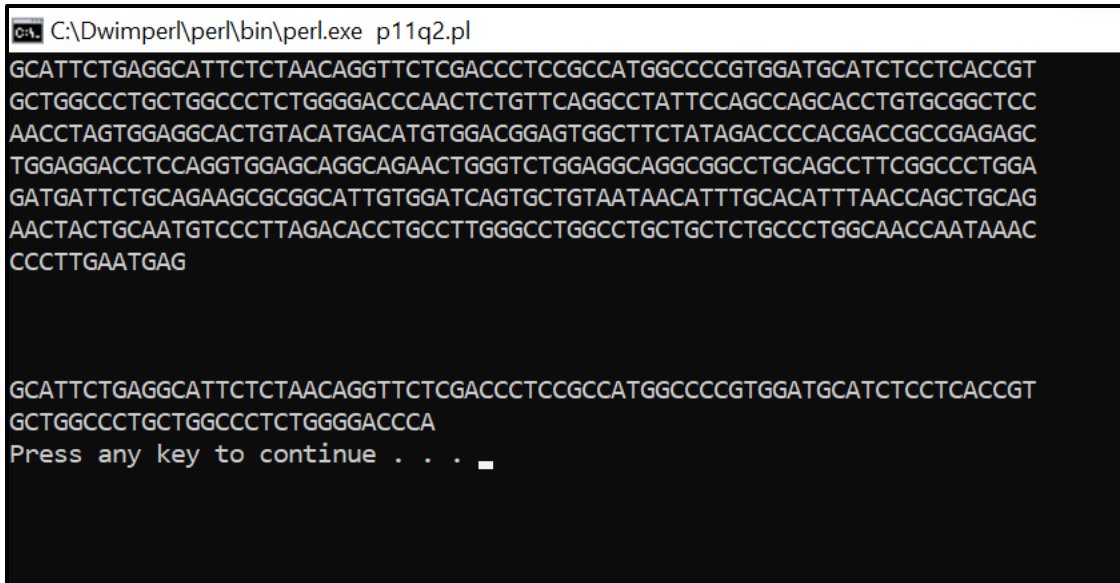
Q.2) Write a Perl script to read the file named as dna.txt and display the sequence as an output using `getc ()` and `read ()` function.

CODE:

```
$file = "C:/Users/lizap/Documents/dna.txt";
open(FH, $file) or die "File can't be opened";
while ($char = getc(FH))
{
    print $char;
}
close(FH);
print "\n\n\n";

$d=" ";
open(file, $file) or die "File can't be opened";
read(file, $d, 100);
print("$d\n");
close(file);
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p11q2.pl
GCATTCTGAGGCATTCTCTAACAGGTTCTCGACCCTCCGCCATGGCCCCGTGGATGCATCTCCTCACCCT
GCTGGCCCTGCTGGCCCTCTGGGGACCCAACCTCTGTTTCAGGCCTATTCCAGCCAGCACCTGTGCGGCTCC
AACCTAGTGGAGGCACCTGTACATGACATGTGGACGGAGTGGCTTCTATAGACCCACGACCGCCGAGAGC
TGGAGGACCTCCAGGTGGAGCAGGCAGAACTGGGTCTGGAGGCAGGCGGCCTGCAGCCTTCGGCCCTGGA
GATGATTCTGCAGAAGCGCGGCATTGTGGATCAGTGCTGTAATAACATTTGCACATTTAACCAGCTGCAG
AACTACTGCAATGTCCCTTAGACACCTGCCTTGGGCCTGGCCTGCTGCTCTGCCCTGGCAACCAATAAAC
CCCTTGAATGAG

GCATTCTGAGGCATTCTCTAACAGGTTCTCGACCCTCCGCCATGGCCCCGTGGATGCATCTCCTCACCCT
GCTGGCCCTGCTGGCCCTCTGGGGACCCA
Press any key to continue . . . _
```

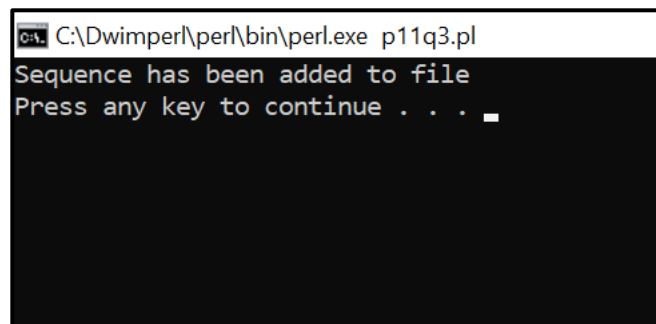
Fig2. Output for a Perl script to read the file named as dna.txt and display the sequence as an output using `getc ()` and `read ()` function

Q.3) Write a Perl script write a sequence i.e. atuauua in file name rna.txt.

CODE:

```
$file = "C:/Users/lizap/Documents/rna.txt";
open(fh, '>', $file) or die "File can't be opened";
$seq = "atuauua";
print fh $seq;
print "Sequence has been added to file\n";
close(fh);
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p11q3.pl
Sequence has been added to file
Press any key to continue . . . _
```

Fig3. Output for a Perl script write a sequence i.e. atuauua in file name rna.txt

Q.4) Write a Perl script for following -

a. makes a new directory

b. Rename a directory

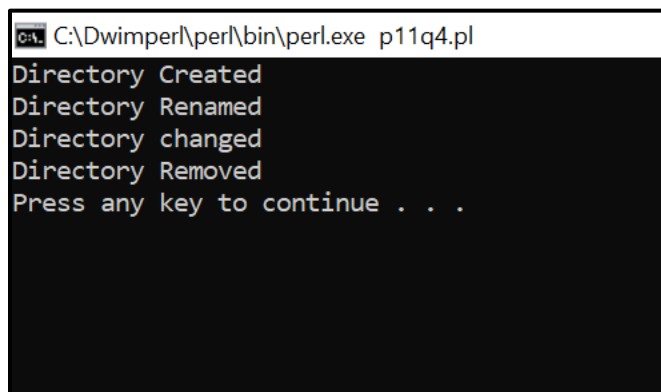
c. changes a directory

d. removes a directory

CODE:

```
$dir = "q4_folder";  
mkdir($dir) or die "Could not make directory";  
print("Directory Created\n");  
  
$dir2 = "q4_folder1";  
rename($dir, $dir2) or die "Could not rename directory";  
print("Directory Renamed\n");  
  
chdir ($dir2) or die "Could not change directory";  
print "Directory changed\n";  
  
chdir "..";  
rmdir ($dir2) or die "Could not remove directory";  
print("Directory Removed\n");
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p11q4.pl  
Directory Created  
Directory Renamed  
Directory changed  
Directory Removed  
Press any key to continue . . .
```

Fig4. Output for a Perl script for making, renaming, changing and removing directory

Q.5) Write a Perl script to demonstrate -d and glob function.

CODE:

```
print("Demonstrating glob function\n");

@files = glob('C:/Users/lizap/OneDrive/Desktop/perl/*.pl');

foreach $n(@files){

    print("$n\n");

}

print("Demonstrating -d function\n");

if(-d('C:/Users/lizap/OneDrive/Desktop/perl')){

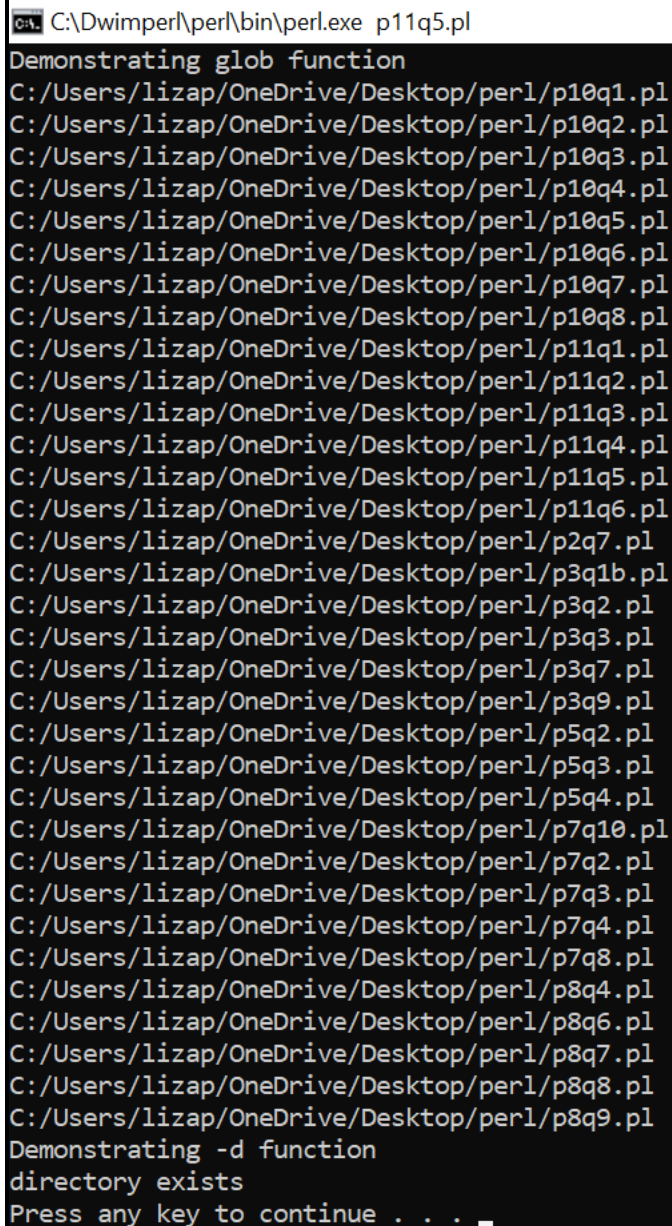
    print("directory exists\n");

}else{

    print("Directory does not exist\n");

}
```

OUTPUT:



```
C:\Dwimper\perl\bin\perl.exe p11q5.pl
Demonstrating glob function
C:/Users/lizap/OneDrive/Desktop/perl/p10q1.pl
C:/Users/lizap/OneDrive/Desktop/perl/p10q2.pl
C:/Users/lizap/OneDrive/Desktop/perl/p10q3.pl
C:/Users/lizap/OneDrive/Desktop/perl/p10q4.pl
C:/Users/lizap/OneDrive/Desktop/perl/p10q5.pl
C:/Users/lizap/OneDrive/Desktop/perl/p10q6.pl
C:/Users/lizap/OneDrive/Desktop/perl/p10q7.pl
C:/Users/lizap/OneDrive/Desktop/perl/p10q8.pl
C:/Users/lizap/OneDrive/Desktop/perl/p11q1.pl
C:/Users/lizap/OneDrive/Desktop/perl/p11q2.pl
C:/Users/lizap/OneDrive/Desktop/perl/p11q3.pl
C:/Users/lizap/OneDrive/Desktop/perl/p11q4.pl
C:/Users/lizap/OneDrive/Desktop/perl/p11q5.pl
C:/Users/lizap/OneDrive/Desktop/perl/p11q6.pl
C:/Users/lizap/OneDrive/Desktop/perl/p2q7.pl
C:/Users/lizap/OneDrive/Desktop/perl/p3q1b.pl
C:/Users/lizap/OneDrive/Desktop/perl/p3q2.pl
C:/Users/lizap/OneDrive/Desktop/perl/p3q3.pl
C:/Users/lizap/OneDrive/Desktop/perl/p3q7.pl
C:/Users/lizap/OneDrive/Desktop/perl/p3q9.pl
C:/Users/lizap/OneDrive/Desktop/perl/p5q2.pl
C:/Users/lizap/OneDrive/Desktop/perl/p5q3.pl
C:/Users/lizap/OneDrive/Desktop/perl/p5q4.pl
C:/Users/lizap/OneDrive/Desktop/perl/p7q10.pl
C:/Users/lizap/OneDrive/Desktop/perl/p7q2.pl
C:/Users/lizap/OneDrive/Desktop/perl/p7q3.pl
C:/Users/lizap/OneDrive/Desktop/perl/p7q4.pl
C:/Users/lizap/OneDrive/Desktop/perl/p7q8.pl
C:/Users/lizap/OneDrive/Desktop/perl/p8q4.pl
C:/Users/lizap/OneDrive/Desktop/perl/p8q6.pl
C:/Users/lizap/OneDrive/Desktop/perl/p8q7.pl
C:/Users/lizap/OneDrive/Desktop/perl/p8q8.pl
C:/Users/lizap/OneDrive/Desktop/perl/p8q9.pl
Demonstrating -d function
directory exists
Press any key to continue . . .
```

Fig5. Output for a Perl script to demonstrate -d and glob function

Q.6) Write a Perl script to open, read and close directory using directory handle.

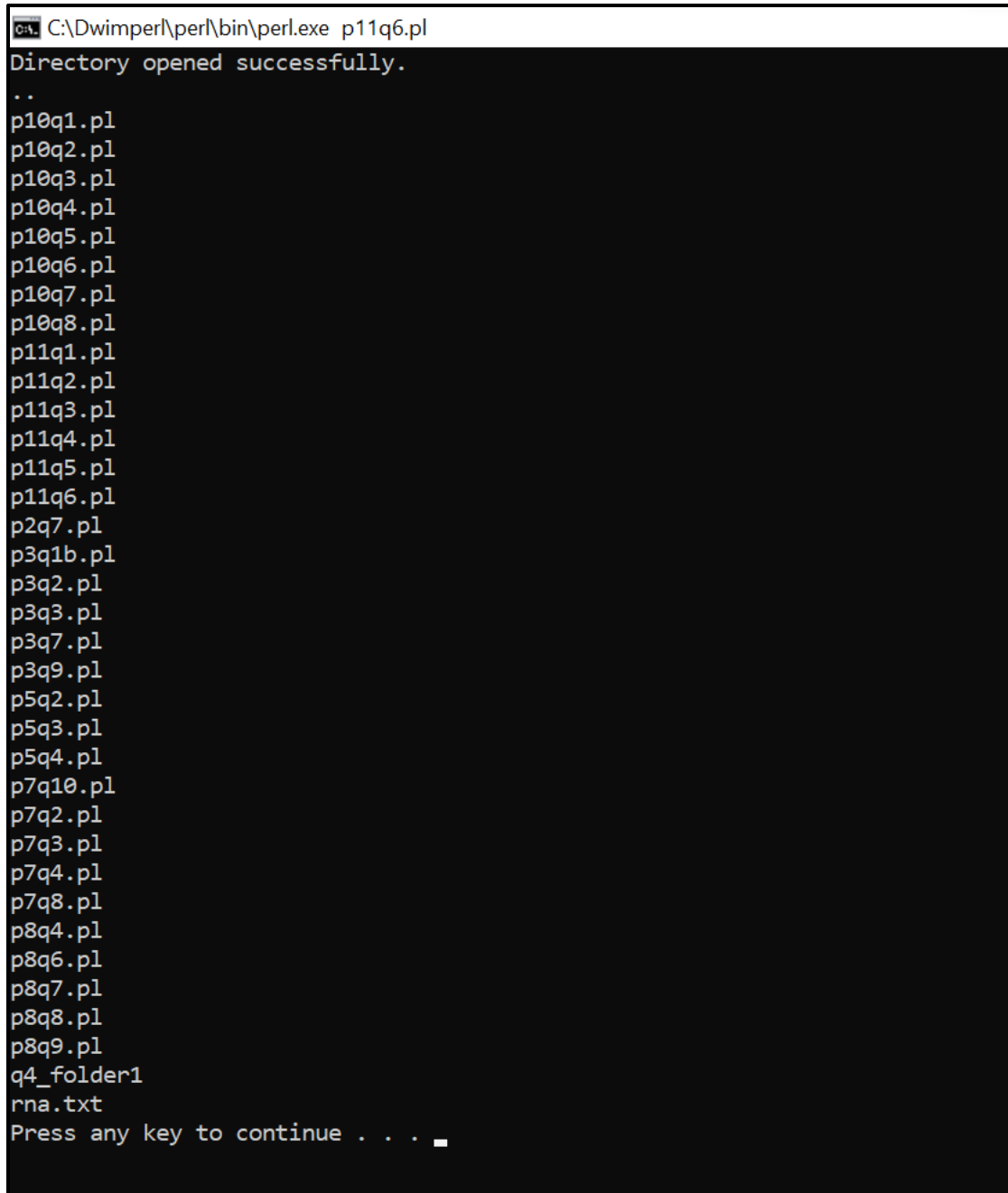
CODE:

```
opendir(NT, 'C:/Users/lizap/OneDrive/Desktop/perl') or die "Could not open directory";
print "Directory opened successfully";
while ($read = readdir(NT))
{
    print "$read\n";
}
```



```
}  
close(NT);
```

OUTPUT:



```
C:\Dwimperl\perl\bin\perl.exe p11q6.pl  
Directory opened successfully.  
..  
p10q1.pl  
p10q2.pl  
p10q3.pl  
p10q4.pl  
p10q5.pl  
p10q6.pl  
p10q7.pl  
p10q8.pl  
p11q1.pl  
p11q2.pl  
p11q3.pl  
p11q4.pl  
p11q5.pl  
p11q6.pl  
p2q7.pl  
p3q1b.pl  
p3q2.pl  
p3q3.pl  
p3q7.pl  
p3q9.pl  
p5q2.pl  
p5q3.pl  
p5q4.pl  
p7q10.pl  
p7q2.pl  
p7q3.pl  
p7q4.pl  
p7q8.pl  
p8q4.pl  
p8q6.pl  
p8q7.pl  
p8q8.pl  
p8q9.pl  
q4_folder1  
rna.txt  
Press any key to continue . . .
```

Fig6. Output for a Perl script to open, read and close directory using directory handle

Practical No. 12

Mongodb Basic commands

AIM:

To understand and write Mongodb basic commands.

THEORY:

MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++. This tutorial will give you great understanding on MongoDB concepts needed to create and deploy a highly scalable and performance-oriented database. MongoDB is a cross-platform, document-oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database - Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection - Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document -A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)
Database Server and Client	
mysqld/Oracle	Mongod
mysql/sqlplus	Mongo

Sample Document - Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

The use Command

MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

Basic syntax of **use DATABASE** statement is as follows –

```
use DATABASE_NAME
```

The dropDatabase() Method

MongoDB **db.dropDatabase()** command is used to drop a existing database.

Syntax

Basic syntax of **dropDatabase()** command is as follows –

```
db.dropDatabase()
```

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

The createCollection() Method

MongoDB **db.createCollection(name, options)** is used to create collection.

Syntax

Basic syntax of **createCollection()** command is as follows –

```
db.createCollection(name, options)
```

In the command, **name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing

Options parameter is optional, so you need to specify only the name of the collection. Following is the list of options you can use –

Field	Type	Description
-------	------	-------------

capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. If you specify true, you need to specify size parameter also.
autoIndexId	Boolean	(Optional) If true, automatically create index on _id field.s Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. If capped is true, then you need to specify this field also.
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

While inserting the document, MongoDB first checks size field of capped collection, then it checks max field.

You can check the created collection by using the command **show collections**.

```
>show collections
```

```
mycollection
```

```
system.indexes
```

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

The drop() Method

MongoDB's **db.collection.drop()** is used to drop a collection from the database.

Syntax

Basic syntax of **drop()** command is as follows –

```
db.COLLECTION_NAME.drop()
```

The insert() Method

To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save()** method.

Syntax

The basic syntax of **insert()** command is as follows –

```
>db.COLLECTION_NAME.insert(document)
```

Here **mycol** is our collection name, as created in the previous chapter. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

The insertOne() method

If you need to insert only one document into a collection you can use this method.

Syntax

The basic syntax of insert() command is as follows –

```
>db.COLLECTION_NAME.insertOne(document)
```

The insertMany() method

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

MongoDB's **update()** and **save()** methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

MongoDB Update() Method

The update() method updates the values in the existing document.

Syntax

The basic syntax of **update()** method is as follows –

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
```

The remove() Method

MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

- **deletion criteria** – (Optional) deletion criteria according to documents will be removed.
- **justOne** – (Optional) if set to true or 1, then remove only one document.

Syntax

Basic syntax of **remove()** method is as follows –

```
>db.COLLECTION_NAME.remove(DELETION_CRITERIA)
```

The Limit() Method

To limit the records in MongoDB, you need to use **limit()** method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

Syntax

The basic syntax of **limit()** method is as follows –

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

Q.1) Create database called Company1 and collection called as Employee with First_Name, Last_Name, Date_Of_Birth, e_mail and phone

Solution:

```
use Company1
```

```
db.createCollection("Employee")
```

a. inserts document into a collection

Solution:

```
db.Employee.insertMany(
```

```
[
```

```
{
```

```
First_Name: "Radhika",
```

```
Last_Name: "Sharma",
```

```
Date_Of_Birth: "1995-09-26",
```

```
e_mail: "radhika_sharma.123@gmail.com",
```

```
phone: "9000012345"
```

```
},
```

```
{
```

```
First_Name: "Rachel",
```

```
Last_Name: "Christopher",
```

```
Date_Of_Birth: "1990-02-16",
```

```
e_mail: "Rachel_Christopher.123@gmail.com",
```

```
phone: "9000054321"
```

```
},
```

```
{
```

```
First_Name: "Fathima",
```

```
Last_Name: "Sheik",
```

```
Date_Of_Birth: "1990-02-16",
```

```
e_mail: "Fathima_Sheik.123@gmail.com",
```

```
phone: "9000054321"
```

```
}
```

```
]
```

```
)
```

b. retrieves the all the document

Solution:

```
db.Employee.find()
```

or

```
db.Employee.find().pretty()
```

c. retrieves the all the document of first two employees

Solution:

```
db.Employee.find().limit(2)
```

or

```
db.Employee.find().pretty().limit(2)
```

d. retrieves the only First Name of first two matching employees but excluding object id

Solution:

```
db.Employee.find({}, {"First_Name":1,_id:0}).pretty().limit(2)
```

e. retrieves the only First Name of employees whose date of birth “1990-02-16” but excluding object id

Solution:

```
db.Employee.find({Date_Of_Birth: "1990-02-16"}, {"First_Name":1,_id:0}).pretty()
```

or

```
db.Employee.find({Date_Of_Birth: "1990-02-16"}, {"First_Name":1,_id:0}).pretty().limit(1)
```

f. retrieves the only First Name of employees whose Last name start with letter ‘C’ but excluding object id

Solution:

```
db.Employee.find({Last_Name: {'$regex': '^C'} }, {"First_Name":1,_id:0}).pretty()
```

g. Update the document whose first name is "Fathima" to "Fathi"

Solution:

```
db.Employee.update({First_Name: "Fathima"},{$set:{ First_Name: "Fathi",}})
```

h. Remove the document whose first name is "Fathima"

Solution:

```
db.Employee.remove({First_Name: "Fathima"})
```

i. drop the collection

Solution:

```
db.Employee.drop();
```

j. drop the database

Solution:

```
db.dropDatabase
```


OUTPUT:

```
Company1> db.Employee.find()
[
  {
    _id: ObjectId("6378eec84988086c10e5b634"),
    First_Name: 'Radhika',
    Last_Name: 'Sharma',
    Date_Of_Birth: '1995-09-26',
    e_mail: 'radhika_sharma.123@gmail.com',
    phone: '9000012345'
  },
  {
    _id: ObjectId("6378eec84988086c10e5b635"),
    First_Name: 'Rachel',
    Last_Name: 'Christopher',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Rachel_Christopher.123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("6378eec84988086c10e5b636"),
    First_Name: 'Fathima',
    Last_Name: 'Sheik',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Fathima_Sheik.123@gmail.com',
    phone: '9000054321'
  }
]
```

```

Company1> db.Employee.find().limit(2)
[
  {
    _id: ObjectId("6378eec84988086c10e5b634"),
    First_Name: 'Radhika',
    Last_Name: 'Sharma',
    Date_Of_Birth: '1995-09-26',
    e_mail: 'radhika_sharma.123@gmail.com',
    phone: '9000012345'
  },
  {
    _id: ObjectId("6378eec84988086c10e5b635"),
    First_Name: 'Rachel',
    Last_Name: 'Christopher',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Rachel_Christopher.123@gmail.com',
    phone: '9000054321'
  }
]
Company1> db.Employee.find({}, {"First_Name":1, _id:0}).pretty().limit(2)
[ { First_Name: 'Radhika' }, { First_Name: 'Rachel' } ]
Company1> db.Employee.find({Date_Of_Birth: "1990-02-16"}, {"First_Name":1, _id:0}).pretty()
[ { First_Name: 'Rachel' }, { First_Name: 'Fathima' } ]
Company1> db.Employee.find({Last_Name: {'$regex': '^C'} }, {"First_Name":1, _id:0}).pretty()
[ { First_Name: 'Rachel' } ]
Company1> db.Employee.update({First_Name: "Fathima"}, {$set: { First_Name: "Fathi", }})
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,

```

```

Company1> db.Employee.remove({First_Name: "Fathima"})
DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDelete, or bulkWrite.
{ acknowledged: true, deletedCount: 0 }
Company1> db.Employee.drop();
true

```

```

Company1> db.dropDatabase();
{ ok: 1, dropped: 'Company1' }
Company1> S|

```

Output for MongoDB commands

Q.2 Create a 'restaurants' collection of restaurant_id, name, address, cuisine, score:

use Hotel;

db.createCollection("restaurants");

db.restaurants.insertMany(

[

{

restaurant_id: "1",

```

name: "WilGokul",
address: "sion",
cuisine: "North Indian",
score: "10"
},
{
restaurant_id: "2",
name: "cesFoo",
address: "sion",
cuisine: "Japanese",
score: "08"
},
{
restaurant_id: "3",
name: "Blabber",
address: "Borivali east",
cuisine: "Lebanese",
score: "10"
}
]
)

```

A. Write a MongoDB query to display all the documents in the collection restaurants.

```
db.restaurants.find();
```

B. Write a MongoDB query to display the fields restaurant_id, name and cuisine for all the documents in the collection restaurant.

```
db.restaurants.find({}, {"restaurant_id" : 1, "name":1, "cuisine" :1});
```

C. Write a MongoDB query to display the fields restaurant_id, name cuisine, but exclude the field _id for all the documents in the collection restaurant.

```
db.restaurants.find({}, {"First_Name":1, "name":1, "cuisine":1, "_id":0});
```

D. Write a MongoDB query to display the fields restaurant_id, name but exclude the field _id for all the documents in the collection restaurant.

```
db.restaurants.find({}, {"First_Name":1, "name":1, _id:0});
```

E. Write a MongoDB query to display all the restaurant which is in the sion address.

```
db.restaurants.find({"address": "sion"});
```

F. Write a MongoDB query to display the first 2 restaurant which is in the sion.

```
db.restaurants.find({"address": "sion"}).limit(2);
```

G. Write a MongoDB query to find the restaurants who achieved a score more than 8.

```
db.restaurants.find({score: { $elemMatch: {"score":{$gt : 8}}}});
```

H. Write a MongoDB query to find the restaurant Id, name and cuisine for those restaurants which contain 'Wil' as first three letters for its name.

```
db.restaurants.find( {name: /^Wil/}, { "restaurant_id" : 1, "name":1, "cuisine" :1 } );
```

I. Write a MongoDB query to find the restaurant Id, name and cuisine for those restaurants which contain 'ces' as last three letters for its name.

```
db.restaurants.find( {name: /^ces/}, { "restaurant_id" : 1, "name":1,"cuisine" :1 } );
```

OUTPUT:

```
Hotel> db.restaurants.find();
[
  {
    _id: ObjectId("6378fd984988086c10e5b637"),
    restaurant_id: '1',
    name: 'WilGokul',
    address: 'sion',
    cuisine: 'North Indian',
    score: '10'
  },
  {
    _id: ObjectId("6378fd984988086c10e5b638"),
    restaurant_id: '2',
    name: 'cesFoo',
    address: 'sion',
    cuisine: 'Japanese',
    score: '08'
  },
  {
    _id: ObjectId("6378fd984988086c10e5b639"),
    restaurant_id: '3',
    name: 'Blabber',
    address: 'Borivali east',
    cuisine: 'Lebanese',
    score: '10'
  }
]
```

```
Hotel> db.restaurants.find({}, {"restaurant_id" : 1, "name":1, "cuisine" :1});
[
  {
    _id: ObjectId("6378fd984988086c10e5b637"),
    restaurant_id: '1',
    name: 'WilGokul',
    cuisine: 'North Indian'
  },
  {
    _id: ObjectId("6378fd984988086c10e5b638"),
    restaurant_id: '2',
    name: 'cesFoo',
    cuisine: 'Japanese'
  },
  {
    _id: ObjectId("6378fd984988086c10e5b639"),
    restaurant_id: '3',
    name: 'Blabber',
    cuisine: 'Lebanese'
  }
]
```

```

Hotel> db.restaurants.find({}, {"First_Name":1, "name":1, "cuisine":1, _id:0});
[
  { name: 'WilGokul', cuisine: 'North Indian' },
  { name: 'cesFoo', cuisine: 'Japanese' },
  { name: 'Blabber', cuisine: 'Lebanese' }
]
Hotel> db.restaurants.find({}, {"First_Name":1, "name":1, _id:0});
[ { name: 'WilGokul' }, { name: 'cesFoo' }, { name: 'Blabber' } ]
Hotel> db.restaurants.find({"address": "sion"});
[
  {
    _id: ObjectId("6378fd984988086c10e5b637"),
    restaurant_id: '1',
    name: 'WilGokul',
    address: 'sion',
    cuisine: 'North Indian',
    score: '10'
  },
  {
    _id: ObjectId("6378fd984988086c10e5b638"),
    restaurant_id: '2',
    name: 'cesFoo',
    address: 'sion',
    cuisine: 'Japanese',
    score: '08'
  }
]

```

```

Hotel> db.restaurants.find({"address": "sion"}).limit(2);
[
  {
    _id: ObjectId("6378fd984988086c10e5b637"),
    restaurant_id: '1',
    name: 'WilGokuL',
    address: 'sion',
    cuisine: 'North Indian',
    score: '10'
  },
  {
    _id: ObjectId("6378fd984988086c10e5b638"),
    restaurant_id: '2',
    name: 'cesFoo',
    address: 'sion',
    cuisine: 'Japanese',
    score: '08'
  }
]
Hotel> db.restaurants.find({score: { $elemMatch: {"score": {$gt : 8}}}});

Hotel> db.restaurants.find( {name: /^Wil/}, { "restaurant_id" : 1, "name":1, "cuisine" :1 } );
[
  {
    _id: ObjectId("6378fd984988086c10e5b637"),
    restaurant_id: '1',
    name: 'WilGokuL',
    cuisine: 'North Indian'
  }
]

```

```

Hotel> db.restaurants.find( {name: /^ces/}, { "restaurant_id" : 1, "name":1,"cuisine" :1 } );
[
  {
    _id: ObjectId("6378fd984988086c10e5b638"),
    restaurant_id: '2',
    name: 'cesFoo',
    cuisine: 'Japanese'
  }
]

```

Output for MongoDB commands