Perl Assignment No. 4

Q1. Define subroutine and call subroutine.

Ans: 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.

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);

In older versions of Perl, the syntax for calling subroutines was slightly different as shown below. This still works in the newest versions of Perl, but it is not recommended since it bypasses the subroutine prototypes.

&subroutine_name(list of arguments);

Q2. Explain passing arguments to subroutine and passing list to subroutine with example.

Ans:

a)Passing arguments to subroutine:-

You can pass various arguments to a subroutine like you do in any other programming language and they can be acessed 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 but passing more than one array or hash normally causes them to lose their separate identities. So we will use references (explained in the next chapter) to pass any array or hash.

```
#!/usr/bin/perl

# Function definition
sub Average {
   # get total number of arguments passed.
   $n = scalar(@_);
   $sum = 0;

   foreach $item (@_) {
      $sum += $item;
   }
   $average = $sum / $n;

   print "Average for the given numbers : $average\n";
}

# Function call
Average(10, 20, 30);
```

When above program is executed, it produces the following result − Average for the given numbers : 20.

b)Passing list to subroutine:-

Because the @_ variable is an array, it can be used to supply lists to a subroutine. However, because of the way in which Perl accepts and parses lists and arrays, it can be difficult to extract the individual elements from @_. If you have to pass a list along with other scalar arguments, then make list as the last argument as shown below –

```
#!/usr/bin/perl

# Function definition
sub PrintList {
   my @list = @_;
   print "Given list is @list\n";
}
$a = 10;
@b = (1, 2, 3, 4);

# Function call with list parameter
PrintList($a, @b);
```

When above program is executed, it produces the following result − Given list is 10 1 2 3 4.

Q3. Explain temporary value via local and state variable via state with example.

Ans: a)Temporary value via local:-The local is mostly used when the current value of a variable must be visible to called subroutines. A local just gives temporary values to global (meaning package) variables. This is known as dynamic scoping. Lexical scoping is done

with my, which works more like C's auto declarations. If more than one variable or expression is given to local, they must be placed in parentheses. This operator works by saving the current values of those variables in its argument list on a hidden stack and restoring them upon exiting the block, subroutine, or eval.

```
#!/usr/bin/perl

# Global variable
$string = "Hello, World!";

sub PrintHello {
   # Private variable for PrintHello function
   local $string;
   $string = "Hello, Perl!";
   PrintMe();
   print "Inside the function PrintHello $string\n";
}
sub PrintMe {
   print "Inside the function PrintMe $string\n";
}

# Function call
PrintHello();
print "Outside the function $string\n";
```

When above program is executed, it produces the following result –

Inside the function PrintMe Hello, Perl!

Inside the function PrintHello Hello, Perl!

Outside the function Hello, World!

b) State variable 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 and available starting from Perl 5.9.4.

```
#!/usr/bin/perl

use feature 'state';

sub PrintCount {
   state $count = 0; # initial value

   print "Value of counter is $count\n";
   $count++;
}

for (1..5) {
   PrintCount();
}
```

When above program is executed, it produces the following result –

Value of counter is

Value of counter is 1

Value of counter is 2

Value of counter is 3

Value of counter is 4

Q4. What is File Handling and opening and closing files?

Ans: 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.

Three basic file handles are - STDIN, STDOUT, and STDERR, which represent standard input, standard output and standard error devices respectively.

Open Function :-

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

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

Here DATA is the file handle, which will be used to read the file.

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";
```

Q5.Write a short note on Chop v/s Chomp.

Ans: Chomp:-

This version of chomp removes any trailing string that corresponds to the current value of '$/' (also known as $INPUT_RECORD_SEPERATER). In the English module it returns the total number of characters remove from all its arguments by default '$/' is set to new line character.

Syntax:

Chomp VARIABLE

chomp (LIST)

chomp

Chop:-

This function removes last character from EXPR. Each element of list $_ if no value is specified.

Syntax:

chop VARIABLE

chop (LIST)

chop

Q6. What is directory in which explain create and remove directory?

Ans:

- A directory is a location for storing files found in hierarchy file system such as Linux, MS-DOS.
- A directory is used to store, organize and separate files and directories on computer.
- For ex. You could have directory to store all pictures and another directories to store all your documents. By storing specific types of file in a folder you could quickly get to the type of file you wanted to view.
- On many computers directories are known as folder, drawer, analogous to workbench or traditional office filing cabinet.

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.

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

# This removes perl directory from /tmp directory.
rmdir( $dir ) or die "Couldn't remove $dir directory, $!";
print "Directory removed successfully\n";
```

Q7. Explain following functions in the File Handling: getc(), read, print, copy files.

Ans:

A) getc function:-

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

```
getc FILEHANDLE
getc
```

B)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.

C)print function:-

For all the different methods used for reading information from filehandles, the main function for writing information back is the print function.

```
print FILEHANDLE LIST
print LIST
print
```

The print function prints the evaluated value of LIST to FILEHANDLE, or to the current output filehandle (STDOUT by default).

D)copy files:-

The Perl copy file is one of the featured and mainly it will handle by using the copy() function and these has one of the module which is used for copied the user input and its contents from one place to another. Here, we use most probably in the file as the intermediate bridge for to string and returning the data from server.

```
#!/usr/bin/perl

# Open file to read
open(DATA1, "<file1.txt");

# Open new file to write
open(DATA2, ">file2.txt");

# Copy data from one file to another.
while(<DATA1>) {
   print DATA2 $_;
}
close( DATA1 );
close( DATA2 );
```