Unit-4

Advanced Perl

① Finer points of looping:

A loop statement allows us to execute a statement or group of statements multiple times while some condition evaluates to true.

1. for loop: Executes a sequence of statements multiple as long as those conditions are met.

```
Eg: for($i=1; $i<=3; $i++)
    {
      print "$i";
    }
```

Output: 123

2. while loop: Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

```
Eg:  $i=3;
     while($i>0)
     {
       print "Hello\n";
       $i--;
     }
```

Output: Hello
Hello
Hello

3. do-while loop: Like a while statement, except that it tests the condition at the end of the loop body.

Ex:
```
$a=5;
do{
  print "$a";
  $a=$a-1;
}while($a>0);
```

output: 5 4 3 2 1

4 foreach loop: The foreach loop iterates over a normal list value and sets variable VAR to be each element of the list in turn.

Ex:
```
@data = ("Red", "Pink", "Blue");
foreach $word(@data)
{
   print "$word\n";
}
```

output:
Red
Pink
Blue

5 until loop: until loop is opposite of while loop. It takes a condition in the parenthesis and it only runs until the condition is false.

Ex:
```
$a=5;
until ($a<1)
{
  print "$a";
  $a=$a-1;
}
```

output:
5 4 3 2 1

Loop Control Statements:

Loop control statements change the execution from its normal sequence.

1. next Statement: Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

2. last statement: Terminates the loop statement and transfers execution to the statement immediately following the loop.

```
$i=1;
En: while ($i<10)
{
  if ($i%2==0)
  {
    $i++;
    next;
  }
  if($i==7)
  { last;
  }
  print "$i ";
  $i++;
}
```

Output:

1 3 5

Infinite Loop: A loop becomes infinite loop if a condition never becomes false. En: for(;;)

```
{
  print "This loop will run forever";
}
```

② Pack and Unpack:

- ~~This~~ The pack function evaluates the expressions in LIST and packs them into a binary structure specified by Expr.
- Each character may be optionally followed by a number, which specifies a repeat count for the type of value being packed, that is, nibbles, chars or even bits, according to the format.
- A value of * repeats for as many values remains in LIST.
- Values can be unpacked with the unpack function.
- For example, a5 indicates that the five letters are expected. b32 indicates that 32 bits are expected. h8 indicates that 8 nibbles are expected, etc.

Syntax: pack Expr, List

- Return value: This function returns a packed version of data in LIST.

usage:

1. a ASCII character string padded with null characters
2. b String of bits
3. c A signed character
4. d double-precision floating point number
5. f single-precision floating-point number
6. i A signed integer
7. x A null byte

* ⇒ [format specified in template]

Ex:

```
$bits = pack("c", 65); #prints A, which is ASCII 65.
print "$bits\n";
$bits = pack("x"); # $bits is now a null character
print "$bits\n";
$bits = pack("sai", 100, "T", 30);
@array = unpack("sai", "$bits"); # Array now contains 3 elements
print "Array $array[0]\n";
print "Array $array[1]\n";
print "Array $array[2]\n";
```

Output:

```
A

100
T
30
```

unpack function:

- This function unpacks the binary string STRING using the format specified in TEMPLATE (a, b, c, d, i, x ...)
- Basically reverses the operation of pack, returning the list of packed values according to supplied format (TEMPLATE)

Syntax:

```
unpack Template, STRING
```

Return value: This function returns the list of unpacked values

③ Perl File Handling:

- File handling is the most important part of any programming language. A filehandle is an internal perl structure that associates with a filename.
- Perl file handling is important as it is helpful in accessing files such as text files, etc.
- Perl file handles are capable of creating, reading, opening and closing a file.

Perl Create file: A file can be created with the help of open() function. The $fh (file handle) is a scalar variable. The '>' sign means we are opening file for writing. The $filename denote the path or file location.

```
my $filename = 'file1.txt';
open (my $fh, '>', $filename) or die "Couldn't open $!";
print $fh "Hello! We have created this file\n";
close $fh;
print "Done\n";
```

output:

```
Done
```

→ Once file is open, use $fh in print statement. The print() function will print the above text in the file.

→ close $fh, closes the file.

Perl Open file:

We can open a file in following ways:

- (<): The < sign is used to open an already existing file. It opens the file in read mode.

Syntax: open FILE, "<", "fileName.txt" or die $!

- (>): The > sign is used to open and create the file if it doesn't exists. It opens the file in write mode.

Syntax: open FILE, ">", "fileName.txt" or die $!

- (+>+<): The < sign will empty the file before opening it. It will clear all the data of that file. To prevent this use (+) sign before ">" or "<" characters.

Syntax: open File, "+<", "filename.txt" or die $!
open File, "+>", "fileName.txt" or die $!

- (>>): The >> sign is used to read and append the file content. It places the file pointer at the end of the file where we can append the information.

Perl Read File:

. We can read complete file at once or we can read it one line at a time. "<" is used to read the file.

Let's assume, we have created a file 'file1.txt' with following content.

This is the first line.
This is the second line.
This is the third line.
This is the fourth line.

To read single line at a time:

First line of the file1.txt will be displayed.

```
my $filename = 'file1.txt';
open(my $fh, "<", $filename) or die "Couldn't open $!";
my $row = <$fh>;
print "$row\n";
print "done\n";
```

Output: This is the first line
done.

To read multi-lines at a time:

```
my $filename = 'file1.txt';
open(my $fh, "<", $filename) or die "Couldn't open $!";
while(my $row = <$fh>)
{
    print "$row\n";
}
print "done\n";
```

Output: This is the first line
This is the second line
continuation
This is the third line
This is the fourth line
done.

Perl Write file:

Through file writing, we'll append lines in the file1.txt. New lines will be added at the last of the file.

```
open(FILE, ">>file1.txt") || die "problem opening $file1.txt\n";
print FILE "this is line is added to file\n";
print FILE @lines1;
print FILE " A complete file is created";
print FILE @lines2;
```

Output: this is line is added to file.
A complete file is created.

Perl close file:

Perl close file is used to close a file handle using close() function. File closing is not compulsory in perl. Perl automatically closes file once the variable is out of scope.

```
open FILE1, "file.txt" or die $!;
...
close FILE1;
```

Perl File Handle Operator:

File handle operator is the main method to read information from a file. It's used to get information/input from user.

Eg:
```
print "What's your age:\n";
$age = <STDIN>;
if ($age >= 18) {
  print "You are eligible to vote\n";
}
else {
  print "You are not eligible to vote\n";
}
```

Output: What's your age?
18
You are eligible to vote.

Perl print() function:

The print() function prints back the information given through filehandle operators.

```
print "Welcome\n";
```

Output: Welcome.

Perl Copying file:

We can copy content of one file into another file as it is...:

```
open (File1Data, "<file1.txt"); #Opening file1 to read.
open (File2Data, ">file2.txt"); # Opening new file to copy
while (<File1Data>)                content of file.
{
  print File2Data $_;
}
close (file1.Data);
close (File2.Data);
```

Renaming a file:

We can rename a file file1.txt to file2.txt. Assuming file is available in /usr/test directory.

```
rename ("/usr/test/file1.txt", "/usr/test/file2.txt");
```

Deleting an existing file:

We can delete an existing file using the 'unlink' function.

```
unlink ("/usr/test/file1.txt");
```

④ Eval:

. The Perl Eval is defined as one of the functions that can be used to evaluate the given user inputs it may be any data types like strings, integer etc.

. The eval function not only validates the strings and user inputs, it also handles the errors and exceptions for both compile and runtime scripts.

. Whatever the input values are used in the eval block, it should be validated once when the script is to be executed.

. Eval block captures the user-defined errors, syntax errors in compile time as well as run-time code snippets.

. Some But, some types of errors are not caught when we use eval function like "uncaught signal errors, Running out of memory in storage areas, user-defined syntax errors" these are compile type errors, so the programmer can identify these errors easily and can be corrected for execution.

Ex: We can use eval() function to trap fatal error.

```
eval { alarm(15) };
warn() if $@;
eval { print ("the print function worked.\n"); };
warn() if $@;
```

. If there is a run-time error, the error is returned in $@.

. Instead of try we use eval and instead of catch we test $@.

⑤ Data Structures:

Perl provides many data structures. There are many kinds of nested data structures.

- Arrays of Array:

. The simplest kind to build is an array of arrays, also called a two dimensional array or a matrix.

. A Creation and Accessing a Two-Dimensional Array:

```
@AoA = ( ["fred", "barn"],
         ["george", "jane", "elliot"],
         ["homer", "Jason", "Harry"],
       );
print $AoA[2][1]; #prints "Jason"
```

. If we want to print the whole thing,

```
for $row (@AoA)
{
  print "$row\n";
}
```

. A two dimensional array can be dynamically created using push operator.

. In order to assign new values to array of arrays, push operator can be used.

Ex: `push @{ $AoA[0] }, "wilma", "betty";`

`# Appending new columns to an existing row`

- Hashes of Arrays:

Declaration:

```
%HoA = (
        fruits => ["freshmelon", "Strawberry"],
        nuts   => ["Cashew", "Almond", "Pista"],
        );
```

- Accessing and printing of Hash of Arrays:

Single element can be accessed using:

```
print $HoA{fruits}[0]; #prints freshmelon
```

- If we want to print the whole thing:

```
for $row (keys %HoA)
{
    print "$row\n";
}
```

- Array of Hashes:

Declaration:

```
@AoH = (
    {
        Lead => "fred";
        Friend => "barney";
    },
    {
        Lead => "george";
        Wife => "janet";
        Son => "elliot";
    },
);
```

- To add key/value to an element:

```
$AoH[0]{pet} = "dino";
$AoH[1]{pet} = "Simba";
```

- Accessing each of the Array of Hashes:

to access one element,

```
print $AoH[0]{lead};    #prints "fred"
```

- Hashes of Hashes:

Declaration:

```
%HoH = (
        Williams => {
            lead => "fred",
            wife => "Georgina",
        },
        Simpsons => {
            lead => "gurge",
            wife => "Caroline",
            son => "James",
        },
    );
```

- Accessing and printing of Hashes of Hashes:

```
print $HoH{Williams}{wife};    #prints "Georgina"
```

⑥ Packages and Modules:

Packages:

- A perl package is a collection of code which resides in its own namespace. Perl module is a package defined in a file having the same name as that of package and having extension .pm.
- The package is declared using "package" keyword.
- Any variable which is not contained in any package belongs to the main package. Therefore all the variables being used, belong to the main package.

Module:

- A module in Perl is a collection of related subroutines and variables that performs a set of programming tasks.
- Perl modules are reusable.

Example: Creation of Perl Module:

A module name must be same as the package name and should end with .pm extension.

Example: Calculator.pm

```
package Calculator;
sub addition      # defining subroutine for addition
{
    $a = $_[0];
    $b = $_[1];      #Initializing variables a & b
```

```
    $a = $a+$b;
    print "Addition is $a\n";
}
sub substraction
{
    $a = $_[0];
    $b = $_[1];
    $a = $a-$b;
    print "Substraction is $a";
}
```

Using Perl module:

```
Use Calculator;        #Package Calculator
print "Enter two numbers to add";
$a = 10;               # defining values to variable
$b = 20;
Calculator::addition($a,$b);  #subroutine call
print "Enter two numbers to subtract";
$a = <STDIN>;
$b = <STDIN>;
Calculator::substraction($a,$b);  #subroutine call.
```

Output: Enter two numbers to add

Addition is 30

Enter two numbers to subtract: 30 10

Substraction is 20

- If a file accessing the package lies outside the directory then we use '::' to tell the path of the module.

⑦ Objects:

- Perl is an object oriented, dynamic and interpreter based programming language.
- In object-oriented programming, there are three main aspects object, class and methods.
- An object is a datatype which can be specifically called as an instance of the class to which it belongs.
- It can be a collection of data variables of different data types as well as collection of different data structures.
- Methods are functions which work on these objects of the class.

First, we need to define the class. In Perl, it's done by building the package of the class.

Ex: package Employee;

Here, Employee is the class name.

Second task, is to create an instance of the package (i.e the object) For this, we need a constructor. A constructor is a subroutine in perl.

Eg: Creation and Implementation of Objects in OOPs.

```
use strict;
use warnings;
package Employee;          # class with name Employee
sub new                    # Constructor with name new
{
  my $class = shift;       #shift will take package name
  my $self = {             # and assign it to variable "class"
              serialNum => shift,
              firstName => shift,
              lastName => shift,
             };
  bless $self, $class      #attaching object with class
  return $self;            #returning instance of class Employee
}
my $object = new Employee (1, "Harry", "Potter");
  print ("$object->{'firstname'}\n");
  print ("$object->{'lastname'}\n");
```

Output:

Harry
Potter

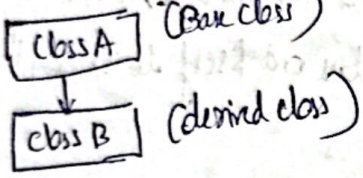
- In the constructor, we are defining a simple hash reference $self to design our object.
- Here, the object will have three values serialNum, firstName and lastName of an Employee.
- The "my" keyword (when used with variable, it makes the variable a local variable) localises $class and $self to be within the enclosed block.
- The 'shift' keyword takes the package name from the default array "@_" and pass it on to the 'bless' function.
- bless function attaches object with class.
- At the end, the constructor will finally return the instance of the class Employee.

Inheritance:

- Inheritance is the ability of any class to extract and use the features of other classes.
- It is the process in which new classes called the derived classes are created from existing or Base classes.
- The basic concept here is that the programmer is able to use the features of one class into another without declaring or defining the same thing again and again in different classes.
- Inheritance is one of the most important features of Object Oriented Programming.

Subclass: The class that inherits properties from another class is called subclass or Derived class.

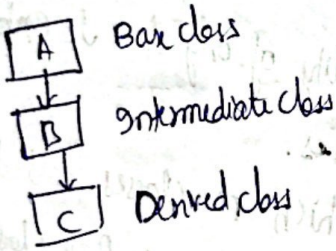
Superclass: The class whose properties are inherited by subclass is called Baseclass or Superclass.



Multilevel Inheritance:

Inheritance in Perl can be of many types but multilevel inheritance is one in which there is a chain of the base class and derived classes.

. In multi-level inheritance, a derived class will be inheriting a base class and as well as the derived class also acts as the base class to other class.



Implementation of Multilevel Inheritance in Perl:

```
-use strict;                #Student class
use warnings;
package student;
```

```
sub new {
    my $class = shift;
    my $self = {
                name => shift,
                roll_no => shift,
               };
    bless $self, $class;
    return $self;
}
```

```
                    # Btech class
package Btech;
use strict;
use warnings;
use parent 'Student';          # Inheriting super class. Student
```

```
use strict;
use warnings;
use Btech;                     # Inheriting Intermediate class
my $a = Btech->new("Sam", 01);  # object creation
print "$a->{'name'}\n";
print "$a->{'roll_no'}\n";
```

output:

```
Sam
1
```

⑧ Interfacing to the operating system:

- Perl is a popular language for system administrators and programmers who have to work with file and directories due to the fact that there are many built-in functions to perform system admin activities. These activities include creating directories, changing the name of files, creating links, and executing programs in the operating system.

- File system calls:

All return a value indicating success or failure.

```
chdir $x or die "cannot chdir to $x\n";
```

Examples of the more common calls for manipulating the file system are:

chdir $x	change directory
unlink $x	delete
rename($x, $y)	rename
rmdir $x	remove directory

- exec:

The exec function terminates the current script and executes the program named as its argument, in the same process.

```
exec "sort $output" or die "can't exec sort\n";
```

• Shell commands:

Perl allows execution of shell commands in two ways, by using the built-in system function or by "quoted execution".

System function: A simple example of system function is

system("date");

The string given as argument is treated as shell command to be run by bin.

Quoted execution: The output of a shell command can be captured using quoted execution.

$date = `date`;

Here, the output of the date command is assigned to variable $date.

• Environmental variables:

The current environmental variables are stored in the special hash %ENV. A script can read them or change them by accessing this hash.

• Process control in Unix:

- Perl implements the classic UNIX process mechanism: the fork function creates a child process as a clone of the current process (the parent process), which runs in parallel with its parent.
- If-then-else construct is used. We use the value returned by the fork function to control if-then-else construct, providing

one block of code to be ~~exte~~ executed in the parent process and another block of code to executed in the child.

- In UNIX paradigm, the first thing the child process does is to use exec to start a new program running.
- A parent can create multiple child processes by repeated use of fork, and a child process can itself create its own children using fork.
- The child process inherits all open filehandles (including pipes) and therefore it is ~~used~~ usual for it to do some ~~tidying~~ tidying up by closing unwanted filehandles.
- A simple example: Suppose we want to create a child process that will send data to its parent, using a pipe (the pipe function sets up pipe, returning two filehandles, one for reading end and one for writing end)

```
pipe(READHANDLE, WRITEHANDLE);
if ($pid = fork)
{  # Parent process
   close WRITEHANDLE;
   ...
} elsif ($pid == 0) {
   # Child process
   close READHANDLE;
   ...
   exit;
} else {  # Something went wrong
   die "Can't fork\n";
}
```

- Process control in Windows:

- A new process is created by calling the constructor.
- The child process runs in parallel with the parent, but is subject to the parent's control.

Ex: $child->suspend();
$child->resume();

- The parent can wait for the child to terminate

$child->wait($timeout);

- If the waiting period is complete without the child process completing, it can be forcibly terminated.

$child->kill($exitcode);

- Controlling OLE Automation Server:

The OLE Automation server is a Windows application which provides an interface by which it can be controlled by other applications.

- The interface takes the form of a collection of objects which have properties and methods.

(9) Creating 'Internet-aware' application:

- The Internet is a rich source of information held on web servers, FTP servers, etc.
- A web browser can access information on web servers and FTP servers, and news servers.
- However, this is not the way of the to the information: an 'internet-aware' application can access a server and collect the information without manual intervention.
- A perl application can establish a connection to the server, send the request in the format that the browser would use, collect the returned HTML and then extract the fields that form the answer to the query.
- In the same way, a perl application can establish a connection to mail server and send request which will result in the server returning a message, listing the no. of currently unread message.
- The LWP (Library for WWW access in Perl) is a collection of modules.
- The LWP::Simple module provides a simple interface to web servers.
- Using LWP::Simple we can retrieve the contents of web page in a single statement.

```
use LWP::Simple
$url = "http://www.study.com/index.html";
$page = get($url);
```

(10) 'Dirty hands' Internet programming!

. Modules like LWP:: simple and LWP:: User Agent meet the needs of most programmers requiring web access, and there are numerous other modules for other types of Internet access.

Eg: Net:: FTP for access to FTP servers.

. Some tasks may require lower level of access to the Internet network, and this is provided by perl both in the form of modules (e.g IO:: Socket) and at an even lower level by built-in functions.

. Access to the internet at this level involves the use of sockets.

. sockets are network communication channels, providing a bi-directional channel between processes on different machines.

. The Socket interface is based on the TCP/IP protocol suite, so that all the information is automatically handled.

. In TCP a reliable channel, with automatic recovery from data loss or corruption: for this reason a TCP connection is often described as a virtual connection.

. The Socket in Perl also permits connections using UDP (User Datagram Protocol)

(ii) Security Issues:

- Language which provides full access to the underlying operating system open up possibilities for malicious actions.
- Such possibilities are even greater when scripts make network access to untrusted sources, and Perl therefore provide an effective way of maintaining control by a process called 'taint' checking.

Taint checking:

- In Perl, taint mode is a way to make the code more secure
- The Perl approach to security is based on the idea that data coming outside is untrustworthy, or 'tainted', and such data coming from outside, cannot be used directly.
- Taint checking is automatically in UNIX environment.
- When taint checking is active, command-line arguments, environment variables and file inputs are marked as tainted.
- Taint mode is used to keep track of data coming from the user and avoids doing anything insecure:

Safe module:

- Taint checking provides security for code written using potentially unsafe data from outside.
- The Safe module allows us to set up 'Safe objects' [unsafe [untrusted code can be executed in isolation from rest of the program]. Thus adding another level of security.