Improvement Test – Nov 2017

Scheme and Solutions

Sub:	UNIX and Shell Programming				Sub Code:	15CS35	Branch:	CSE/ISE
Date:	21-11-2017	Duration:	90 min's	Max Marks: 50	Sem / Sec:	ISE (3A,B)		OBE

Q.1 a) Explain following commands with examples
i) nice ii) cron iii) nohup iv) kill
Each command with syntax and explanation-2.5*4=10M
1M –Explanation
1.5M-Syntax
nice:
nice runs command with an adjusted "niceness", which affects process scheduling. A process with a lower niceness value is given higher priority and more CPU time. A process with a higher niceness value (a "nicer" process) is given a lower priority and less CPU time, freeing up resources for processes that are more demanding.

nice –n 5 sort myfile.txt

ii) cron
cron is a daemon process.It executes the programs at regular intervals.
The cron (short for "cron table") is a list of commands that are scheduled to run at regular time intervals on your computer system.The crontab file will be kept in /var/spool/cron/cron/crontabs

minute 00 through 59 Number of minutes after the hour

hour 00 through 23 (midnight is 00)

day-of-month 01 through 31

month-of-year 01 through 12

day-of-week 01 through 07 (Monday is 01, Sunday is 07)

The first five fields are time option fields. You must specify all five of these fields an asterisk (*) in a field if you want to ignore that field.

Examples:

00-10 17 * 3.6.9.12 5 find / -newer .last_time –print >backuplist

In the above entry, the find command will be executed every minute in the first 10 minutes after 5 p.m. every Friday of the months March, June, September and December of every year.

iii) nohup- Log out safely

The name **nohup** stands for "no hangup." The hangup (**HUP**) signal, which is normally sent to a process to inform it that the user has logged off (or "hung up"), is intercepted by **nohup**, allowing the process to continue running.

The syntax is as follows

```
nohup command-name &
$nohop sort emp.lst &
586
Sending output to nohup.out

$ps –f –u kumar
UID  PID  PPID  C  STIME  TTY  TIME  CMD
KUMAR 586 1     45 14:52   01   0:13  sort emp.lst
```

iv) kill – Termination of a process

The default signal for **kill** is TERM (which will terminate or "kill" the process). Use **-l** or **-L** to list available signals. Particularly useful signals include HUP, INT, KILL, STOP, CONT, and **0**. Alternate signals may be specified in three ways: **-9**, **-SIGKILL** or **-KILL**.

```
$kill 105
```
Kills a process having pid 105

```
$sort –o emp.lst emp.lst &
345
$kill $!

$kill –s kill 121
```

Q.2 a) Explain the mechanism of process creation
Process creation 1M
Fork-1M
Exec -1M
Wait -1M

MECHANISM OF PROCESS CREATION

There are three distinct phases in the creation of a process and uses three important system calls or functions they are namely **fork exec and wait**

Fork :

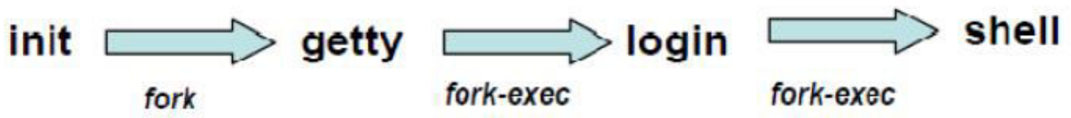
- **A process in UNIX is created with the fork system call**, which creates a copy of the process that invokes it.
- In this mechanism the calling process makes a call to the system call fork(), which then makes an exact copy of itself. The copy will be of the memory of the calling process at the time of the fork() system call. The process image is identical to that of the calling process, except for a few parameters like the PID.
- The child gets a new PID. Forking process is responsible for the multiplication of processes in the system.

Exec:

- Forking creates a process but it is not enough run a new program.
- The forked child overwrites its own image with the code and data of the new program. This mechanism is called exec, and the child process is said to *exec* a new program, using one of the family of exec system calls.
- The PID and PPID of the exec'd process remain unchanged.
- At the end of the execution, a call is made to the exit() function that terminates the child and sends a signal back to the parent after which, the parent becomes free to continue with its other functions.

Wait:

- The parent then executes the wait system call to *wait* for the child process to complete its task.
- Later parent process picks up the exit status of the child (terminate) and continues with its other functions.
- Note that a parent need not decide to wait for the child to terminate.



Q. 2 b) Explain find command with its options
Usage of find command-1M
Any 5 option with explanation 1M*5=5M

find : locating files

It recursively **examines a directory tree to look for files matching some criteria, and then takes some action on the selected files.**

Syntax: **find path_list selecton_criteria action**

- First, it Recursively examines all files in the directories specified in path_list
- It then matches each file for one or more selection-criteria
- Finally, It takes some action on those selected files

The **path_list** comprises one or more directories separated by whitespace. There can also be a host of **selection_criteria** that you can use to match a file, and multiple **actions** to dispose of the file.

Example:

find command to locate all files named a.out

```
$find / -name a.out -print
/home/kumar/scripts/a.out

/home/user2/script/cprogram/a.out

/home/user/a.out
```

```
find . -name “*.c” –print          // All files with extension .c

find . -name '[A-Z]*' –print       // All files begin with an upper case letter
```

Selection Criteria

Locating a File by Inode Number (-inum) : Based on the specified inode number files are searched.

```
find / -inum 13975 -print  2> /dev/null

find: cannot read dir /usr/lost+found: perm

/usr/bin/gzip

/usr/bin/gunzip
```

File type and Permission (-type and -perm) : The –**type** option indicates which type of file. The letter f,d or l selects files of the ordinary, directory and symbolic link type.

```
find . -type d -print 2>/dev/null
```

The –**perm** option specifies the permissions to match. –perm 666 selects files having read and write permissions for all categories of users.

```
find $HOME -perm 777 -type d -print
```

Command to list all that have been modified **since less last 2 days:**

```
find . -mtime -2 -print
```

find /home -atime +365 -print // files that are not been accesses for more than a year

The find operators (!, -o , -a)

- There are 3 operators that are commonly used with the find.
- **!** operator is used before an option to **negate its meaning.**
- **find . ! -name "*.c" –print** *// Finds all files excluding .c file.*
- **-o operator which represent an OR condition.**
- **find /home \(-name "*.sh " -o -name "*.pl" \) -print** ***// here it searches files with .sh or .pl extensions.***
- **The –a operator represents the AND condition**, and is implied by default whenever two selection criteria are placed together.

Q.3 a) Explain the following string handling functions in PERL with examples
i) length ii) index iii) reverse iv) substr v) splice
Each function 2M*5=10M

STRING HANDLING FUNCTIONS

- ➢ **length** determines the length of its argument.

- ✓ $x =“ abcdijklm”;
- ✓ print **length($x);**

➢ **index(s1, s2)** determines the position of a string s2 within string s1.

- ✓ $x =“ abcdijklm”;
- ✓ print **index($x,j);**

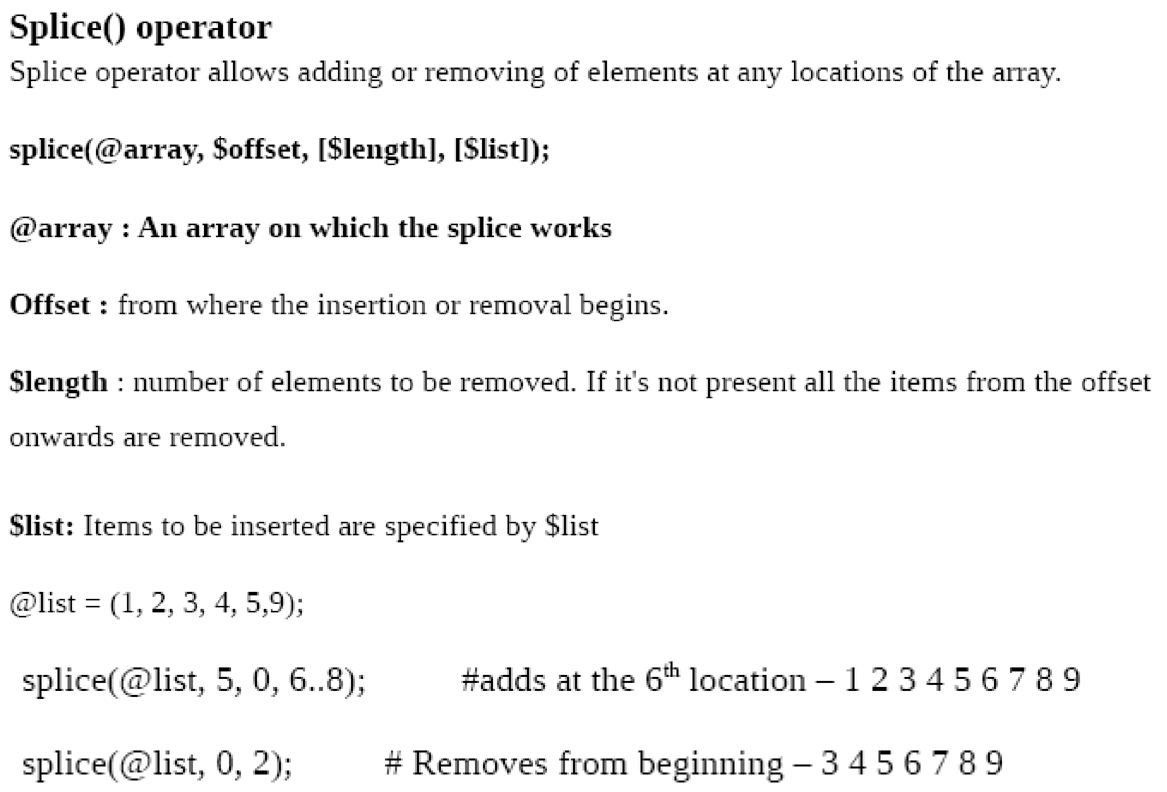
➢ **substr(str,m,n)** extracts a substring from a string str, m represents the starting point of extraction and n indicates the number of characters to be extracted. Substr can extract characters from the right of the string, and insert or replace a string.

- ✓ **substr($x,4,0)** =“efgh”;   // stuffs the string$x with efgh without replacing any string, 0 indicates non replacement
- ✓ print “$x”;   # abcdefghijklm
- ✓ $y = **substr($x,-3,2);**   // extracts 2 characters from the third position on the right
- ✓ print “$y”   # $y is kl

➢ **reverse(str)** reverses the characters contained in string str

- ✓ $x =“hello”;
- ✓ print **reverse($x);**   # print olleh

Explain the following in PERL with examples



Splice() operator

Splice operator allows adding or removing of elements at any locations of the array.

splice(@array, $offset, [$length], [$list]);

@array : An array on which the splice works

Offset : from where the insertion or removal begins.

$length : number of elements to be removed. If it's not present all the items from the offset onwards are removed.

$list: Items to be inserted are specified by $list

```
@list = (1, 2, 3, 4, 5,9);

splice(@list, 5, 0, 6..8);          #adds at the 6th location – 1 2 3 4 5 6 7 8 9

splice(@list, 0, 2);          # Removes from beginning – 3 4 5 6 7 8 9
```

Q. 4 a) Explain the following in PERL with examples

i) for each loop ii) join iii) split
for loop 4M
join 3M
split 3M

<u>Splitting into an array :</u>Split result will be stored in an array.

When split function is used and the returned values are not stored in an array explicitly, they will be stored in the special array @_ by implication. Ex: split(/:/);

When no string is mentioned explicitly, the split function works on the default variable $_. Ex: @COLOR = split(/:/);

When no field separator is mentioned explicitly white spaces are taken as the field separator by default. Ex: split();

@fields = split(/:/, "1:2:3:4:5");

print "Field values are: @fields\n";

result:

Field values are: 1 2 3 4 5

Join: joining a list

- Combines its arguments into a single string and uses the delimiter as the first argument. The remaining arguments could be either an array name or a list of variables or string to be joined.

- **$result = join(“ ”, “this”, “is”,”an”,”example”);** //$result = this is an example

- **$week = join(“ “ , “mon”,”tue”,”wed”);**

- **print $week   # mon tue wed**

foreach: Looping Through a List

- foreach construct is used to loop through a list. It is also used to execute the set of statements repeatedly.

```
foreach $var (@arr) {

statements

}
```

- Each element of the array @arr is picked up and assigned to the variable $var.
- The iteration continued as many items as there are items in the list.

EXAMPLE #!/usr/bin/perl

```
@list = ("10", "20", "30", "40", "50", "60");

print("Here are the numbers in the list: \n");

foreach $temp (@list) {

print("$temp ");

}
```

Q.5 a) Explain with suitable example push and pop functions in Perl
Push 2M
Pop 2M

Perl supports operations both at the left and right of an array.

push() and pop() functions

Push takes first argument as an array variable name and second is the element to be pushed. **push** adds the element to the **end of the array.**

```
@list = (1,2,3,4);

push(@list,5);                    # now the list contents are 1 2 3 4 5
```

pop and shift operator is used to remove elements from an array. It works with only one argument- an array variable name.

pop(@list);   # Removes the last element 0 1 2 3 4

Q.5 b) Explain file handling in Perl with example
File handle definition 1M
Open 2M
Read and write mode 2M
Close 1M

<u>**FILE HANDLE**</u>

- File handle is a program variable that acts as a reference between perl program and the operating system's file structure.
- File handle names do not begin with the special characters and digits & which preferably uses uppercase letters.
- Streams are default file handles which are referred as STDIN, STDOUT and STDERR. STDIN connects to keyboard and STDOUT,STDERR connect to display screen.
- **NULL filehandle :** Allows scripts to get input from either STDIN or from each file listed on the command line.
- It is written as <>, and is called Angle operator, diamond operator or line-reading operator.

<u>**open()**</u>

File is opened using the open() function

Syntax: open(FILEHANDLE, “>,<<,< filename”);

Character(s)	**Meaning**
<	Input(default) i.e. Read Only Access
>	Output, starting at the beginning of the file (Creates, Writes, and Overwrite)
>>	Output starting at the end of the existing data on the file(Writes, Appends)
+>	Input from and output to the file
+<	Read, processed and rewritten to the same file, replacing the data that was red.

Example: **open(INFILE, “<emp.lst”);**

Opens the file emp.lst in read only format.

File can be opened in a append mode as : **open (OUTFILE, “>>pgm.lst”);**

Perl script to open two files, one for read and another for write.

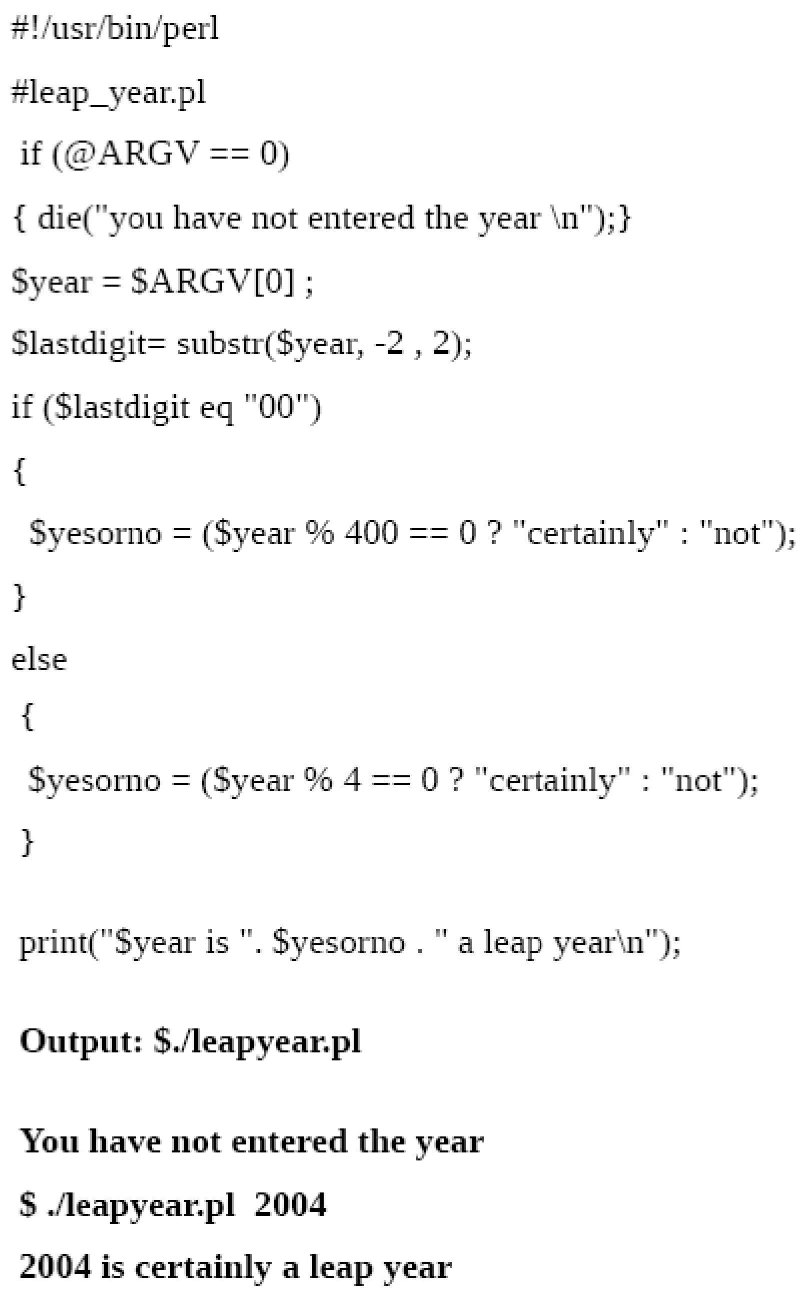
```
#!/usr/bin/perl
open(INFILE, "answer.txt") || die("Cannot open file");
open(OUTFILE, ">write.txt");
while(<INFILE>) {
print OUTFILE if(1..3);
}
close(INFILE);
close(OUTFILE);
```

close()

Closes the file which was already opened.

```
close(FILEHANDLE);
```

Q. 6 a) Using **command line arguments**, write a PERL program to find whether a given year is leap year



```
#!/usr/bin/perl
#leap_year.pl
 if (@ARGV == 0)
{ die("you have not entered the year \n");}
$year = $ARGV[0] ;
$lastdigit= substr($year, -2 , 2);
if ($lastdigit eq "00")
{
  $yesorno = ($year % 400 == 0 ? "certainly" : "not");
}
else
 {
  $yesorno = ($year % 4 == 0 ? "certainly" : "not");
 }

 print("$year is ". $yesorno . " a leap year\n");
```

Output:
```
$./leapyear.pl

You have not entered the year
$ ./leapyear.pl  2004
2004 is certainly a leap year
```

Q. 6 b) Write a Perl script to copy contents of one file to another

```
#!/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 );
```

Q. 7a) Explain lists and arrays in PERL with examples in detail

List definition and example 1+2M=3M
Array definition +example + operation=1+2M+4M=7M

A list is a collection of scalar values enclosed in parentheses.

Elements in a list are comma separated and entire collection of scalars is enclosed within parentheses. The following is a simple example of a list:

(1, 5.3, "hello", 2);

Assigning values to the elements of a list : Values to the elements of a list can be assigned individually to every scalar variable element of the list one by one or at a single stretch by using the syntax of the statement below:

($branch, $year, $sem) = (“cs”, “2015”, “6”);

When the above statement is executed $branch will get the value cs, $year will get the value 2015 and $sem will get the value 6 respectively.

Arrays

The lists can be assigned to special variables known as array variables

An array is a variable that holds multiple values in series. Array names begin with @.

Arrays are the placeholders of lists, which is created by assigning a list to it.

Examples:

@subject = ("maths", "co", "ADE");

In the above example subject is the array name, and 3 values are assigned to it.

@month[1,3..5,12]=(“jan”,“feb”,”mar”,”may”,”jun”);

Last index of the array is referred by $#. $# is also used to set the array to a specific size or delete all its element.

```
$#month = 10;          # array size is set to 11
$#month = -1;          #no elements, elements are deleted
```

Length of the array is determined as : $length = @month;

Reading a file into an array

```
@line = <>; // reads entire line from command line
Print @line; // contains lines of the file
```

Example :

```
#!/usr/bin/perl
@days_betwn = ("wed","thu");
@days=(mon,tue,@days_betwn,fri);
@days[5,6]=qw/sat sun/;
$length=@days;
print ("the third day $days[2]\n");
print("the days of week @days\n");
print("number of elemets $length\n");
print("last subscript of the array $#days\n");
```

Output: ./ar.pl

```
The third day wed
The days of the week mon,tue,wed,thu,fri,sat,sun
The number of elements in the array is 7
The last subscript of the array is 6
```

Q. 8a) Explain the default variable $. and $_
$_=2M
$.=2M

Perl assigns the **line read from input** to a special variable, $_, often called the **default variable.**

$_, represents the last line read or last pattern matched. It holds the current line. Many functions use $_ as a default argument when no argument is mentioned explicitly.

Examples : Normally print function will expect variables, or a list or a string, the value of which is expected to be printed. If no argument is specified for print function it will print the value of default variable $_.

```
$_ = “Hello good morning\n”;
print;
```

$. is used to store the current line number. It is used to represent a line address and to select lines from anywhere. Following code segment prints the message line number is 10 as soon as the current line number ($.) becomes 10.

Example 1: if ($. = 10)

```
{
    print “line number is $.\n”
}
```

Example 2:’ perl –ne ‘print if ($. < 4)’ in.dat   # is similar to head –n 3 in.dat

Q. 8b) Write PERL script to convert a decimal number to binary.

```
#!/usr/bin/perl

foreach $num (@ARGV) {

$temp = $num;

until ($num == 0) { // the condition is false and keep executing that till the conditio
becomes true. then terminates

$bit = $num % 2;

unshift(@bit_arr, $bit);

$num = int($num/2);

}

$binary_num = join(“ ”,@bit_arr);

print (“Binary form of $temp is $binary_num\n”);

$#bit_arr = -1;

}
```

```
Output: $./dec2bin.pl 2 7 6 5

Binary form of 2 is 010

Binary form of 7 is 111

Binary form of 65 is  1000001
```