SET-224 /CET-225
Operating Systems

LAB # 05

LAB Title

Focusing on the usage of Iteration statements and Functions in Shell Scripts.

Assessment of CLO: 04, PLO: 03

Student Name:			
Roll No.			
Semester		Session	

S. No.	Perf. Level / Criteria	Excellent (2.5)	Good (2)	Satisfactory (1.5)	Needs Improvement (0 ~ 1)	Marks Obtained
1	Project Execution & Implementation	Fully functional, optimized, and well-structured.	Minor errors, mostly functional.	Some errors, requires guidance.	Major errors, non-functional, or not Performed.	
2	Results & Debugging Or Troubleshooting	Accurate results with effective debugging Or Troubleshooting.	Mostly correct, some debugging Or Troubleshooting needed.	Partial results, minimal debugging Or Troubleshooting.	Incorrect results, no debugging Or Troubleshooting, or not attempted.	
3	Problem-Solving & Adaptability (VIVA)	Creative approach, efficiently solves challenges.	Adapts well, minor struggles.	Some adaptability, needs guidance.	Lacks innovation or no innovation, unable to solve problems.	
4	Report Quality & Documentation	Clear, structured, with detailed visuals.	Mostly clear, minor gaps.	Some clarity issues, missing details.	Poorly structured, lacks clarity, or not submitted.	
					Total Marks Obtained Out of 10	

Experiment evaluated by

Instructor's Name	Engr.Bushra Aziz		
Date		Signature	

Objective: Focusing on the usage of Iteration statements and Functions in Shell Scripts.

Theory:

The shell languages also provide several iteration or looping statements. The most commonly used of these is the "for statement".

The for Statement

The 'for' statement executes the commands that are contained within it a specified number of times. The 'bash' and 'pdksh' have two variations of the for statement.

The first form of the for statement that bash and pdksh support has the following syntax:

```
for var1 in list do commands
done
```

In this form, the for statement executes once for each item in the list. This list can be a variable that contains several words separated by spaces, or it can be a list of values that is typed directly into the statement. Each time through the loop, the variable var1 is assigned the current item in the list, until the last one is reached.

The second form of for statement has the following syntax:

```
for var1 do statements
done
```

In this form, the for statement executes once for each item in the variable var1. When this syntax of the for statement is used, the shell program assumes that the var1 variable contains all the positional parameters that were passed in to the shell program on the command line.

Typically, this form of for statement is the equivalent of writing the following for statement:

```
for var1 in "$@"
do statements
done
```

The equivalent of the for statement in tcsh is called the foreach statement. It behaves in the same manner as the bash and pdksh for statement. The syntax of the foreach statement is the following:

```
foreach name (list)
 commands
end
```

The while Statement

Another iteration statement offered by the shell programming language is the while statement. This statement causes a block of code to be executed while a provided conditional expression is true. The syntax for the while statement in bash and pdksh is the following:

```
while expression do statements

done
```

The syntax for the while statement in tcsh is the following:

```
while (expression) statements

end
```

Care must be taken with the while statements because the loop will never terminate if the specifies condition never evaluates to false.

BASH while Loop Example

```
#!/bin/bash

b=1

while [ $b -le 10 ]

do

echo "$b"

b=$((b+1))

done
```

The break Statement

The break statement can be used to terminate an iteration loop, such as a for, until or repeat command.

```
#!/bin/bash

#breaking a loop

num=1

while [ $num –lt 10 ] do

if [ $num –eq 5 ] then break

fi

done

echo "Loop is complete"
```

The Continue Statement

Continue is a command which is used to skip the current iteration in for, while, and until loop. It is used in scripting languages and shell scripts to control the flow of executions. It takes one more parameter [N], if N is mentioned then it continues from the nth enclosing loop.

```
#/bin/bash
for a in 1 2 3 4 5 6 7 8 9 10
do
# if a = 5 then continue the loop and
# don't move to line 8
if [ $a == 5 ]
then
continue
fi
echo "Iteration no $a"
done
```

The exit Statement

The exit statement can be used to exit a shell program. A number can be optionally used after exit. If the current shell program has been called by another shell program, the calling program can check for the code and make a decision accordingly.

Functions in shell scripts:

The shell languages enable you to define your own functions. These functions behave in much the same way as functions you define in C or other programming languages. The main advantage of using functions as opposed to writing all of your shell code in line is for organizational purposes. Code written using functions tends to be much easier to read and maintain and also tends to be smaller, because you can group common code into functions instead of putting it everywhere it is needed. The syntax for creating a function in bash and pdksh is the following:

```
fname () { shell commands
}
```

pdksh also allows the following syntax:

```
function fname { shell commands
}
```

Both of these forms behave in the exact same way. Once you have defined your function using one of these forms, you can invoke it by entering the following command:

fname [parm1 parm2 parm3 ...]

Shell functions have their own command line argument.

Use variable $1, $2..$n to access argument passed to the function.

The syntax is as follows:

```
name(){   arg1=$1   arg2=$2   command on $arg1
}
```

To invoke the function, use the following syntax:

```
name foo bar
```

Where,

name = function name.

foo = Argument # 1 passed to the function (positional parameter # 1).

bar = Argument # 2 passed to the function.

EXERCISES

1. Write a script that calculates the average of all even numbers less than or equal to your roll number and prints the result.
2. Write scripts that **Create a function called factorial that:**

- Accepts one parameter (a number)
- Calculates its factorial using a while or for loop
- Returns/prints the result