CS29206 Systems Programming Laboratory, Spring 2022–2023

Class Test 2

13–April–2023 03:00pm–04:00pm Maximum marks: 60

Roll no: ____________________ **Name:** ____

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions.*]

1. (a) Consider a text file **`input.txt`**, containing alphanumeric text along with special characters. Write a single grep command which matches one (or both) of the following two kinds of strings.

(i) Strings starting with a lower-case letter at the beginning of the line, followed by any number of alphanumeric characters, and ending with a lower-case vowel (not necessarily at the end of the line),

(ii) Strings starting with an upper-case letter (not necessarily at the beginning of a line), followed by any number of characters (alphanumeric or special), and ending with a lower-case letter at the end of the line. **(5)**

```
grep -e '^[a-z][a-zA-Z0-9]*[aeiou]' -e '[A-Z].*[a-z]$' input.txt
```

(b) Write an executable gawk *script* which reads a string **`S`** as input from the user (use **`getline`**, do not read from a database), containing multiple fields delimited by **`;`** as the separator. Each field of **`S`** contains an alphanumeric string. Assume that **`S`** contains no spaces. For instance, the user may enter **`One;two;three;67tt7;852`** as **`S`**. Your script should print *Success* if the first or the last field of the input **`S`** is a numeric string (an integer in decimal notation, without any **+** or **−** sign); otherwise it should print *Failure*. Write a function **`compare()`** with suitable argument(s) (say, the string **`S`**) for solving this matching problem. However, the script should read the string **`S`** and print the message (*Success* or *Failure*) outside the function. Note that **`twelve`** and **`5six7`** are not numeric strings, whereas **`12`** and **`567`** are. Also assume that the empty string is not a numeric string. **(10)**

```
#!/usr/bin/gawk -f

function compare (s)
{
   n = split(s, a, ";");
   if ( (a[1] ~ /^[0-9][0-9]*$/) || (a[n] ~ /^[0-9][0-9]*$/) )
      status = "Success"
}

BEGIN {
   status = "Failure"
   getline S < "-"
   compare(S)
   print status
}
```

2. (a) Consider entries in a telephone directory with filename **`directory.txt`** as follows.

```
+123-334-889-778
+880-1855-456-907
+9-7777-38644-808
+123-443-998-887
```

Write a gawk *command* (not a script) which takes **`directory.txt`** as a command-line argument, and the prints only the country codes in all the lines, as shown below. The same country code may be printed multiple times.

```
+123
+880
+9
+123
```

(3)

```
gawk -F- '{ print $1 }' directory.txt
```

(b) Consider a student dossier file which contains student names in a class and their respective native states. A dossier file from Prof. Artim is given to the right. The file starts with a header line, and is followed by actual student data. The name and the state fields may contain spaces, and are separated by a comma (no space just before or just after the comma). The header may be different in different files. For example, Prof. Foostein's file has the header **`Name,Bundesland`**, whereas Prof. Barbouki's file has the header **`Nom,Région`**.

Apply the notion of associative arrays, and write an executable gawk script to print all the states of the students appearing in a given dossier file. Each state appearing in the file (like **`West Bengal`** in Prof. Artim's file) should be printed only once, and the state header (like **`State`** or **`Bundesland`** or **`Région`**) must not be printed. There is no need to write the student names against every state.

```
Student Name,State
Bar Yash Foorole,Maharashtra
Foolan Barik,West Bengal
Rabin,Bihar
Swetha V V V Y,Karnataka
Naveen Praveen Reddy,Telangana
Samir Sengupta,West Bengal
Sundar F. B.,Uttar Pradesh
Rab Oof,West Bengal
Lalitha,Karnataka
Lolita,West Bengal
Venu Murali Vamshi,Tamil Nadu
Barendra Salam,Manipur
```

(12)

In the code given below, the associative array **`state[]`** is accessed against every state found in the dossier file. If you choose, you can set that entry to anything like **`0`** or **`1`** or **`"found"`**.

```
#!/usr/bin/gawk -f

BEGIN { FS = "," }

{ if (NR > 1) state[$2] }

END { for (s in state) print s }
```

3. **(4 × 5)**

(a) Write a sequence of bash commands in a script, that reads (from the terminal) two parts of the user's name (may contain spaces) in variables **`firstname`** and **`lastname`**, and then sets and prints a variable **`fullname`**. Write your answer below the following sample I/O. The quotes shown should be printed.

```
First name: Foo Bar
Last name: Basu Roy Chowdhury
Full name: "Foo Bar Basu Roy Chowdhury"
```

```
echo -n "First name: "
read firstname
echo -n "Last name: "
read lastname
fullname="$firstname $lastname"
echo "Full name: \"$fullname\""
```

(b) What will be printed by the following bash code snippet? Write your answer below the snippet.

```
declare -ai P=(2 3 5 7)
P[5]=11
echo ${P[3]}
echo ${P[@]}
echo ${!P[@]}
echo ${#P[@]}
```

```
7
2 3 5 7 11
0 1 2 3 5
5
```

(c) Suppose that the bash variable **`pattern`** stores a regular expression. You want to search for this regular expression in a file **`myfile.txt`**. You can use the following command:

```
grep -e "$pattern" myfile.txt
```

Two other methods for the same search are sketched below. The first method uses pipe **`|`**, and the second method uses string redirection **`<<<`**. Fill in the blanks (write nowhere else) to complete the commands of these alternative methods. In each blank, use a standard Unix command to print the entire file **`myfile.txt`** to **`stdout`**.

```
                    cat myfile.txt                     | grep -e "$pattern"

grep -e "$pattern" <<<                    `cat myfile.txt`
```

(d) What will be printed by the following bash code snippet? Write your answer below the snippet.

```
x=15; y=25
function Fxy () {
   echo "x = $x, y = $y"
}
function F () {
   Fxy
   y=30; local x=10 y=20
   Fxy
}
F
Fxy
```

```
x = 15, y = 25
x = 10, y = 20
x = 15, y = 30
```

(e) What will be printed by the following bash code snippet? Write your answer below the snippet.

```
function f () {
    echo echo Hello, World!
}
g=`f`
echo "$g"
echo '$g'
echo `$g`
`echo $g`
```

```
echo Hello, World!
$g
Hello, World!
Hello, World!
```

4. Write an executable bash script to do the following task. The script uses a directory name **`dir`**. If that name is supplied by the user as the first command-line parameter, **`dir`** is set to that parameter, otherwise **`dir`** is set to the current directory. The script then checks whether **`dir`** is a directory and has read permission. If not, it exits with some error status. Otherwise, it proceeds to create a file **`myfiles.zip`** in the current directory as outlined below (assume that you have permission to write in the current directory). The script checks whether **`myfiles.zip`** is already present in the current directory, and if that is the case, the script deletes the file. After that, the zip file is created using the following command, where **`file1`**, **`file2`**, **`file3`**, . . . are all of the regular files in **`dir`** having read permission.

```
zip myfiles.zip file1 file2 file3 ...
```

Before invoking the command, the script makes a listing of all the files in **`dir`**, and identifies (and stores) the names of all the regular files in **`dir`** with read permission. If there is no such file, the **`zip`** command is not invoked. Otherwise, **`myfiles.zip`** is created using the above command. Write the executable bash script below to perform this task. Note that you should call **`zip`** only once (provided that there are file(s) in **`dir`** to zip). You must not incrementally add file(s) to the zip archive. **(10)**

```
#!/bin/bash

if [ $# -eq 0 ]; then dir="."; else dir="$1"; fi
if [ ! -d $dir ]; then echo "$dir is not a directory"; exit 1; fi
if [ ! -r $dir ]; then echo "$dir is not readable"; exit 2; fi
echo "Going to zip files in the directory \"$dir\""
zipfile=myfiles.zip
if [ -e $zipfile ]; then rm $zipfile; fi
flist=""
declare -i n=0
for file in `ls $dir`; do
   if [ -f "$dir/$file" ] && [ -r "$dir/$file" ]; then
      flist+=" $dir/$file"
      n=$((n+1))
   fi
done
if [ $n -eq 0 ]; then
   echo "There are no files to zip"
else
   echo "Going to zip the following $n files"
   echo $flist
   zip $zipfile $flist
fi
```