Prepared by: E. Bedalli & I. Draçi

Deadline: 30.05.2022

CEN308 –Operating Systems

Homework 1

Problem 1

Prepare a shell script (in BASH) named gcd.sh that receives two command line arguments which are natural numbers and prints as result the greatest common divisor.
Sample execution:

```
./gcd.sh 20 16
Gcd(20, 16)=4
```

Solution:

The most simplistic algorithm I know to solve the greatest common divider is the Euclidian Algorithm. The steps are:

- assign maximum and minimum number.
- until maximum % minimum == 0 start a loop.
- make minimum = maximum % minimum. (always 0 to minimum-1 so it is smaller than minimum)
- make maximum = minimum.
- end loop.
- print minimum which will be the greatest common divider.

The only trick here is to know how to swap with a temporary value the values and how to do a while loop in Bash Scripting.

Done by: Edison Ponari

Prepared by: E. Bedalli & I. Draçi

Deadline: 30.05.2022

Problem 2

The final round of a famous TV show was the "Split or Steal" round. Here, each of the two contestants would secretly pick among options "Split" or "Steal" for the jackpot amount that was collected during the previous rounds. Then:

- If both contestants have choosen "Split", they each receive half of the jackpot.
- If one chooses "Steal" and the other chooses "Split", the "Steal" contestant wins the entire jackpot and the "Split" contestant leaves with nothing.
- If both choose "Steal", neither contestant wins any money.

Prepare a shell script (in BASH) named splitorsteal.sh which receives as input argument the jackpot amount.

Then the choices of the two contestants are generated randomly, and finally the amount of money won by each contestant is printed on the console.

Then the choices of the two contestants are read, and finally the amount of money won by each contestant. No input validation is required, assume that the input will be regular.

Sample execution:

```
./splitorsteal.sh 250000
First contestant chose: split
Second contestant chose: split
The first contestant won 125000 and the second contestant won 125000.
```

Another sample:

```
./splitorsteal.sh 32000
First contestant chose: split
Second contestant chose: steal
The first contestant won 0 and the second contestant won 32000.
```

Another sample:

```
./splitorsteal.sh 65000
First contestant chose: steal
Second contestant chose: steal
The first contestant won 0 and the second contestant won 0.
```

Solution:

If you know that 2 competitors gave the same answer, they will win the same amount (0 or half). Otherwise, the competitor that chooses to steal gets the overall sum and the other gets 0. Expressing these 4 scenarios in Bash Scripting was easy using if else statements. To get the user input we can use the read -p call to prompt the user to enter a value and save them into variables. Also integer casting is important to use when splitting the sum in half.

Done by: Edison Ponari

Prepared by: E. Bedalli & I. Draçi

Deadline: 30.05.2022

Problem 3

Write a bash script named guess.sh that generates a random number between 1 and 10 (both included) and you have to try and guess it.
If your guess is less than/greater than enter your guess: 9generated number, then print on the console your guess is (less then | greater then) the secret.
If you guessed the secret number, then print the total number of tries before guessing it.
Sample:

```
./guess.sh
enter your guess: 9
your guess is less than the secret number
enter your guess: 10
you guessed it after 2 guesses
```

Solution:

Bash Scripting has a until loop that will repeat instructions until the condition is fulfilled. Using this, each time you take input from the user as the guess and compare it to the secret number. If it is the same you print the number of tries the user did.

Problem 4

Prepare a shell script (in BASH) named getInfo.sh that receives one command line argument that is the name of a file or directory (in the working directory). If the argument is a file, then print the size of the file in bytes. If it is a directory, then list directory content. In both cases you should print whether the user has read, write and execute permission on the file or directory
If the argument doesn't exist, then create a new file with that name and write the current date to the file content.

Solution:

Using the -d and -f flag, we can understand if the argument of the user is a file or a directory. The files size can be checked using stat command while ls can be used to see the files a directory. For the permissions on a file, we use ls -l to check permissions on write, read and execute. If the argument isn't either a file or directory, we create a file using touch command and write to it using >> operator. Date of the system is called by date command.

Done by: Edison Ponari

Problem 5

Prepare a shell script (in BASH) named listArgs.sh that receives one or more command line arguments that are expected to represent names of a file or directories. Then the script should print as output the list of arguments that are directories and the list of arguments that are files. Note: the arguments which are neither files not folders will not appear in any of the lists.
Sample execution:

```
./ listArgs.sh A B C D E F Z
The list of files: A B C
The list of directories: D E F
```

Solution:

Using shift command, we can go from argument to argument. Using this and a while loop, for each file we can check using -d and -f for files and directory. They are saved in two strings: files and directories. When the $1, first argument, is empty, we finished with the loop and print the two strings with concatenated file names and directories names.

Problem 6

Prepare a bash script called logRotation.sh that prompts the user to enter a file name. If the input file doesn't exist, exit the script, otherwise check if the size of the given file is more than 1 MB, copy the content of the file into a new file with the same file name + current date + file extension and empty the content of the given file.

Solution:

Using the command to see the size we did in one of the previous exercises, we can check if the size is less than 1024*1024 bytes. If it is, we just copy the file into a new name. To get the extension for the name, we use the ${file#*.} format. The date was used in previous exercises too, so it wasn't a problem.

Problem 7

Prepare a shell script (in BASH) named copyLimited.sh that receives two command line arguments: a name of a subdirectory and a positive integer N. Firstly it will check if the current subdirectory doesn't exist, and in such case it will be created. Then it will make a copy (with the same name) for each .txt file of the current directory into the specified

Done by: Edison Ponari

subdirectory but if .txt file contains more than N rows, then the copy will contain only the first N rows of the original file.
(Hint: consider the command wc –l with piping or redirection)

Solution:

The only difference from the previous exercise was that we needed a command to count lines and also a command to get a specific number of lines. To get the number of lines, we use this command $(wc -l < $file) and if it is bigger than N, we need only the first N lines. To get the first N lines, we can use this command for splitting head -n $n $file. After this they are saved in the new directory with the same names.

Problem 8

Prepare a shell script (in BASH) named commonFiles.sh that receives two or three command line arguments: the first and second argument are the names of two subdirectories and the third command line argument (which is optional) is the extension (like pdf, txt, py etc.). If the script is executed with two command line arguments, then it will provide a list of all files which exist in both directories (i.e. having the same name, the contents may not be the same). If the script is executed with three command line arguments, then it will provide a list of all files of the given extension which exist in both directories (i.e. having the same name, the contents may not be the same).
Sample execution:

```
./commonFiles.sh
THE LIST OF COMMON FILES:
manual.pdf
organization.jpg
report.pdf
sample.txt
solution.py
work.txt
```

Another sample:

```
./commonFiles.sh py
THE LIST OF COMMON FILES:
solution.py
```

Solution:

We can iterate over the first folder and for each file search if it exists in the other folder. If there is a third argument, the loop will not look like *.* but *.$3 since the extension of the files should be like the

third argument. For each match, we concatenate the filename to a string called common. In the end we just print the common string containing all file names that these 2 directories have in common.

Problem 9

Assume that teaching assistant needs a shell script named submissionsReport.sh which will be run in a working directory containing several subdirectories each of them named by the ID of a student who has made a submission. Inside the folder of each student there should be the solutions of the homework exercises which have to be named precisely: hw1.c, hw2.c, hw3.c … and so on. If the script is run without providing any argument, the default number of exercises is 5, otherwise the number of exercises is specified by the first (and only) command line argument of the script. Then the script must print for each student a message of the form: Student <id> has submitted <nr_solutions> solutions: <list of exercises which are submitted by the student>.

Sample execution:

```
./submissionReport.sh
Student I0015A has submitted 3 solution(s): 1 3 4
Student I0016B has submitted 2 solution(s): 2 3
Student I0020A has submitted 4 solution(s): 1 2 3 4
Student I0027B has submitted 1 solution(s): 2
```

Solution:

N is the number of homework each student should have. If there is an argument in the command execution, we assign it to N which as a initial value of 5. In the current directory, we loop over each directory that we have. To do this, we use the ./* pattern and use an if statement with the -d to get only the directories. For each directory, we check using a seq n for hw1.c – hwn.c using hw$item.c pattern. After seeing all hw files, we print the number of homework kept track by a counter and the list of them concatenated.

Problem 10

GitHub is a famous platform for code version control and collaboration. One of the most interesting features is showing technologies a project uses according to the file extensions and their memory. Create a shell script that when executed in a folder containing different types of files, it organizes it in folders according to the extension. Afterwards it shows a table with the disk usage for each type of file.

Solution:

Iterating through each file, get the extension and if there isn't a directory with that name, create one. After that, move the file to that directory. After this, for each directory get the size and get the size for each folder. Using division, give the percentage for each folder compared to the total size.