OPERATING SYSTEMS – FINAL PROJECT

Enkel Shehdula, Jord Thanasi

Part I

Provide a well-documented report with the following parts , both in MS Word and HTML with step by step analysis and explanations in tutorial fashion.

1) Running and Analyzing Scripts

Examine the following scripts. Run them, then shortly explain them. You should experiment with different inputs, provide the relevant results through screenshots, after extensively commenting the given codes. You should also, provide the intermediate, not only the final results of each script, by adding "echo" commands whenever needed to show intermediate results and how each script is behaved in detail, so that to support your analysis. Therefore, the analysis of each script will not be only additional comments, in the already existing ones, but, also, step by step intermediate results proving your comments.

Example A-1. *mailformat*: Formatting an e-mail message

This Bash script, mail_format.sh, is designed to clean and format email message files by removing unnecessary characters and improving readability. It first checks if exactly one argument (a valid filename) is provided; otherwise, it exits with an error. If the file exists, it uses a predefined sed script to strip out leading > characters (common in quoted emails), remove leading and trailing whitespace, and eliminate tabs. The cleaned content is then piped into the fold command, which wraps lines to a maximum width of 70 characters, ensuring that long lines are broken at word boundaries for better readability. This script serves as a lightweight, efficient alternative to bulky formatting tools by leveraging powerful Unix text utilities.

```
#!/bin/bash
# mail_format.sh (ver. 1.1): Format email messages.
# Gets rid of carets, tabs, and also folds excessively long lines.
# =================================================================
# Standard Check for Script Argument(s)
ARGS=1    #Declare that script expects only 1 argument (FileName)
E_BADARGS=85   #Error Code to exit when wrong # of args are entered
E_NOFILE=86    #Error Code to exit when filename wrong

if [ $# -ne $ARGS ]; then # Correct number of arguments passed to script?
  echo "Usage: `basename $0` filename"   #Output to help user understand how to
run script
  exit $E_BADARGS
fi

if [ -f "$1" ]; then # Check if file exists.
  file_name=$1    #Saves Filename from argument to variable file_name
else
  echo "File \"$1\" does not exist."
  exit $E_NOFILE
fi

# -----------------------------------------------------------------
MAXWIDTH=70 # Width to fold excessively long lines to.
# ================================
# A variable can hold a sed script.
# It's a useful technique.
sedscript='s/^>//   #First Line removes all carates , tabs and extensive long
spaces The code in the Project file produced outputs so we formated it
s/^ *>//
s/^[[:space:]]*//
s/[[:space:]]*$//'
# ================================
# Delete carets and tabs at beginning of lines,
#+ then fold lines to $MAXWIDTH characters.
sed "$sedscript" "$file_name" | fold -s --w=$MAXWIDTH
#The above command executes the sedscript with the file as parameter then pipes
the
#output to the fold command to fold the text with the specficied max width
# -s option to "fold" breaks lines at whitespace, if possible.
```

```
# This script was inspired by an article in a well-known trade journal
#+ extolling a 164K MS Windows utility with similar functionality.
#
# A nice set of text processing utilities and an efficient
#+ scripting language provide an alternative to the bloated executables
#+ of a clunky operating system.
exit $?  #exit
```

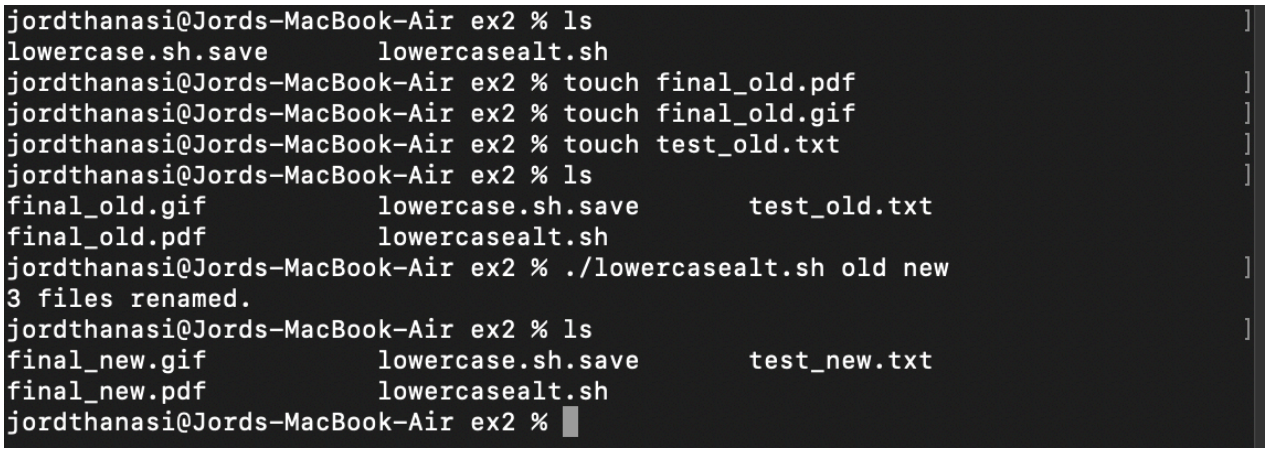
```
jordthanasi@Jords-MacBook-Air ex1 % ls
mail_format.sh  mail_test.txt
jordthanasi@Jords-MacBook-Air ex1 % cat email_test.txt
cat: email_test.txt: No such file or directory
jordthanasi@Jords-MacBook-Air ex1 % cat mail_test.txt
>   This is a line in the email
>>     This is a long email line that should be formatted and wrapped after 70 c
haracters or so.
   Another line with too many spaces and tabs   %
jordthanasi@Jords-MacBook-Air ex1 % ./mail_format.sh mail_test.txt
This is a line in the email
This is a long email line that should be formatted and wrapped after
70 characters or so.
Another line with too many spaces and tabs%
jordthanasi@Jords-MacBook-Air ex1 %
```

Example A-2. *rn*: A simple-minded file renaming utility

This script is a modification of Example 16-22.

This Bash script, rn.sh, is a simple file renaming utility that replaces a specified text pattern in filenames within the current directory. It expects two arguments: the pattern to find and the replacement string. If the correct number of arguments isn't provided, it displays usage instructions and exits. The script loops through all files whose names contain the old pattern, and for each match, it uses sed to replace the old pattern with the new one in the filename, then renames the file using mv. It keeps a count of how many files were renamed and prints a grammatically correct message based on the count.

```
#! /bin/bash
# rn.sh
# Very simpleminded filename "rename" utility (based on "lowercase.sh").
#
# The "ren" utility, by Vladimir Lanin (lanin@csd2.nyu.edu),
#+ does a much better job of this.
ARGS=2 #Script expects 2 arguments (old format)(new format)
E_BADARGS=85 #Exit status for wrong arg count
ONE=1 # For getting singular/plural right (see below).
if [ $# -ne "$ARGS" ] #checks number of arguments entered
then
echo "Usage: `basename $0` old-pattern new-pattern" #Print usage in case of
wrong arg counter

# As in "rn gif jpg", which renames all gif files in working directory to
jpg.
exit $E_BADARGS
fi
number=0 # Keeps track of how many files actually renamed.Later to be
incremented
for filename in *$1* #Traverse all matching files in directory.
do
if [ -f "$filename" ] # If finds match...
then
fname=`basename $filename` # Strip off path.
n=`echo $fname | sed -e "s/$1/$2/"` # Substitute new for old in filename.
mv $fname $n # Rename.
let "number += 1" #Increment every for every file transformed
fi #End if
done #End loop
if [ "$number" -eq "$ONE" ] # For correct grammar.
then
echo "$number file renamed." #echo-ed if only 1 file renamed
else
echo "$number files renamed." #echo-ed if  +1 files renamed
fi #End if
exit $? #Exit
# Exercises:
# ---------
# What types of files will this not work on?
# How can this be fixed?
```



This script won't handle filenames with spaces or special characters correctly and may overwrite files without warning. It could be improved by quoting variables (e.g., "$fname"), adding checks for file name collisions, and using more robust pattern matching.

Example A-3. *blank-rename*: Renames filenames containing

blanks This is an even simpler-minded version of the previous script.

This Bash script, rename_blank.sh, scans all files in the current directory and rename any file that contains spaces in its name by replacing those spaces with underscores (_). It initializes counters and flags to track how many files are renamed. For each file, it checks if the filename contains a space using grep -q " ". If it does, it uses sed to substitute all spaces with underscores and then renames the file using mv. It keeps a count of the renamed files and, at the end, prints a grammatically correct message depending on whether one or multiple files were renamed. This script helps clean up filenames that can cause issues in shell scripts or command-line tools due to unescaped spaces.

```
#! /bin/bash
# blank-rename.sh
#
# Substitutes underscores for blanks in all the filenames in a directory.
ONE=1 # For getting singular/plural right (see below).For changing echo
statements in the final if condition based on # of files renamed
number=0 # Keeps track of how many files actually renamed.Counter
FOUND=0 # Successful return value.Used as a flag that is true when a file with
space in name is found
for filename in * #Traverse all files in directory.
do
echo "$filename" | grep -q " " # Pipes the name of the current file to the grep
to check  if the file name has spaces
if [ $? -eq $FOUND ] # Flag = true if file has space in name
then
fname=$filename # Assign current file name in a temporary variable fname

n=`echo $fname | sed -e "s/ /_/g"` # Pipe name of the current file in the
sedscript to subsitute space with underscore
# and save the new name on another temporary variable n.
mv "$fname" "$n" # Change the name of the current file with the new name
let "number += 1" #Increment the counter of the renamed files
fi #If Statement end
done #Loop ending
if [ "$number" -eq "$ONE" ] #If statement to change the echo statements based
on # of files renamed.
then
echo "$number file renamed." #Only 1 file renamed
else
echo "$number files renamed." #+1 files renamed
fi # End if
exit 0 #End script
```

```
jordthanasi@Jords-MacBook-Air ex3 % touch "file 1.txt"
jordthanasi@Jords-MacBook-Air ex3 % touch "file 2.pdf" "file 3.gif" "file 4.txt"
 normal.txt
jordthanasi@Jords-MacBook-Air ex3 % ls
file 1.txt      file 3.gif      normal.txt
file 2.pdf      file 4.txt      rename_blank.sh
jordthanasi@Jords-MacBook-Air ex3 % ./rename_blank.sh
4 files renamed.
jordthanasi@Jords-MacBook-Air ex3 % ls
file_1.txt      file_3.gif      normal.txt
file_2.pdf      file_4.txt      rename_blank.sh
jordthanasi@Jords-MacBook-Air ex3 %
```

Example A-4. *encryptedpw*: Uploading to an ftp site, using a locally encrypted password

This Bash script (ex72..sh) is designed to securely upload a file to a remote server using FTP with an encrypted password. It begins by checking if a filename argument is provided; if not, it exits with an error code. The script then sets up key variables: Username for the login name, pword pointing to a file containing an encrypted password, and Filename, which extracts the base name of the file to upload. The encrypted password is decrypted using a utility called cruft, which implements a one-time pad encryption method. The FTP session is then started using the ftp command with the -n option to disable auto-login. Within the FTP session block, the script logs in using the username and decrypted password, switches to binary mode, navigates to the target directory on the remote server, uploads the specified file, and ends the session. Although encryption is used for storing the password, the script notes that transmitting it in plain text (as FTP does) is still insecure, and recommends using ssh for sensitive operations.

```
#!/bin/bash
# Example "ex72.sh" modified to use encrypted password.
# Note that this is still rather insecure,
#+ since the decrypted password is sent in the clear.
# Use something like "ssh" if this is a concern.
E_BADARGS=85
if [ -z "$1" ]
then
echo "Usage: `basename $0` filename"
exit $E_BADARGS
fi
Username=example # Change to suit.
pword=/home/bozo/secret/password_encrypted.file
# File containing encrypted password.
Filename=`basename $1` # Strips pathname out of file name.
Server="XXX"
Directory="YYY" # Change above to actual server name & directory.
Password=`cruft <$pword` # Decrypt password.
# Uses the author's own "cruft" file encryption package,
#+ based on the classic "onetime pad" algorithm,
#+ and obtainable from:
#+ Primary-site: ftp://ibiblio.org/pub/Linux/utils/file
#+ cruft-0.2.tar.gz [16k]
ftp -n $Server <<End-Of-Session
user $Username $Password

binary
bell
cd $Directory
put $Filename
bye
End-Of-Session
# -n option to "ftp" disables auto-logon.
# Note that "bell" rings 'bell' after each file transfer.
exit 0
```

```
[jordthanasi@Jords-MacBook-Air ex4 % ls                                        ]
ex4.sh
[jordthanasi@Jords-MacBook-Air ex4 % ./ex4.sh                                  ]
Usage: ex4.sh filename
[jordthanasi@Jords-MacBook-Air ex4 % touch file.txt                            ]
[jordthanasi@Jords-MacBook-Air ex4 % ls                                        ]
ex4.sh          file.txt
[jordthanasi@Jords-MacBook-Air ex4 % ./ex4.sh file.txt                         ]
./ex4.sh: line 18: /home/example/secret/password_encrypted.file: No such file or
 directory
ftp: xxx: nodename nor servname provided, or not known
Not connected.
Not connected.
Bell mode on.
Not connected.
Not connected.
jordthanasi@Jords-MacBook-Air ex4 %
```

Example A-5. *copy-cd*: Copying a data CD

This Bash script, copy-cd.sh, automates the process of copying a data CD to an ISO image and then burning that image onto a blank CD-R. It first defines the CD-ROM device (/dev/cdrom), the output ISO file path, a block size for copying (2048 bytes, standard for CDs), and the CD writing device identifier (used by wodim). It prompts the user to insert the source CD (without mounting it), then uses dd to make a raw copy of the CD to an ISO file. After copying, it asks the user to remove the original CD and insert a blank CD-R, then uses wodim (or cdrecord on some systems) to burn the ISO to the new disc. Finally, it offers the user the option to delete the large ISO file and performs the action based on their input. This script is useful for backing up or duplicating data CDs using command-line tools.

```
#!/bin/bash
# copy-cd.sh: copying a data CD

# Define the CD-ROM device and output file paths
CDROM=/dev/cdrom # CD ROM device
OF=/home/bozo/projects/cdimage.iso # output file
# Change the output file path to suit your system.

# Set the block size for copying
BLOCKSIZE=2048

# DEVICE variable for the cdrecord/wodim command
DEVICE="1,0,0"

# Prompt user to insert the source CD and press ENTER
echo; echo "Insert source CD, but do *not* mount it."
echo "Press ENTER when ready."
read ready # Wait for input, $ready not used.

# Notify user that the CD is being copied to the output file
echo; echo "Copying the source CD to $OF."
echo "This may take a while. Please be patient."

# Use dd to copy the CD-ROM to the output file with the specified block size
dd if=$CDROM of=$OF bs=$BLOCKSIZE # Raw device copy.

# Prompt user to remove the source CD and insert a blank CD-R
echo; echo "Remove data CD."
echo "Insert blank CDR."
echo "Press ENTER when ready."
read ready # Wait for input, $ready not used.

# Notify user that the output file is being copied to the blank CD-R
echo "Copying $OF to CDR."

# Use wodim to burn the ISO file to the blank CD-R
wodim -v -isosize dev=$DEVICE $OF
# Uses Joerg Schilling's "cdrecord" package (see its docs).
# http://www.fokus.gmd.de/nthp/employees/schilling/cdrecord.html
# Newer Linux distros may use "wodim" rather than "cdrecord" ...

# Notify user that the copying process is complete
echo; echo "Done copying $OF to CDR on device $CDROM."
```

```
# Ask user if they want to erase the ISO file
echo "Do you want to erase the image file (y/n)? " # Probably a huge file.
read answer

# Case statement to handle user input for erasing the ISO file
case "$answer" in
  [yY]|[yY][eE][sS]) # Accepts 'y', 'Y', 'yes', and 'Yes' as input
    rm -f $OF
    echo "$OF erased."
    ;;
  *)
    echo "$OF not erased."
    ;;
esac

# End of script
echo
exit 0
```

```
jordthanasi@Jords-MacBook-Air ex5 % nano ex5.sh
jordthanasi@Jords-MacBook-Air ex5 % chmod +x ex5.sh
jordthanasi@Jords-MacBook-Air ex5 % ls
ex5.sh
jordthanasi@Jords-MacBook-Air ex5 % ./ex5.sh

Insert source CD, but do *not* mount it.
Press ENTER when ready.


Copying the source CD to /home/example/projects/cdimage.iso.
This may take a while. Please be patient.
dd: /dev/cdrom: No such file or directory

Remove data CD.
Insert blank CDR.
Press ENTER when ready.

Copying /home/example/projects/cdimage.iso to CDR.
./ex5.sh: line 37: wodim: command not found

Done copying /home/example/projects/cdimage.iso to CDR on device /dev/cdrom.
Do you want to erase the image file (y/n)?
n
/home/example/projects/cdimage.iso not erased.

jordthanasi@Jords-MacBook-Air ex5 %
```

Example A-6. Collatz series

This Bash script, collatz.sh, implements the Collatz sequence (also known as the hailstone sequence), a famous mathematical conjecture that transforms any positive integer into a sequence that always ends in the loop 4 → 2 → 1. The script accepts a seed number as a command-line argument; if none is provided, it uses the process ID ($$) as a fallback. It then performs up to MAX_ITERATIONS (200 by default) steps, printing the current number at each iteration. If the current number is even, it divides it by 2; if odd, it multiplies it by 3 and adds 1. Each number is neatly printed using printf in a column format, with 10 numbers per line. The result is a visually organized sequence showing how the number fluctuates before (presumably) reaching the repeating 4, 2, 1 cycle. This script showcases iterative feedback (the output becoming the next input) and is a classic example of chaotic yet eventually stable behavior in simple math.

```
#!/bin/bash
# collatz.sh
# The notorious "hailstone" or Collatz series.
# -------------------------------------------
# 1) Get the integer "seed" from the command-line.
# 2) NUMBER <-- seed
# 3) Print NUMBER.
# 4) If NUMBER is even, divide by 2, or
# 5)+ if odd, multiply by 3 and add 1.
# 6) NUMBER <-- result
# 7) Loop back to step 3 (for specified number of iterations).
#
# The theory is that every such sequence,
#+ no matter how large the initial value,
#+ eventually settles down to repeating "4,2,1..." cycles,
#+ even after fluctuating through a wide range of values.
#
# This is an instance of an "iterate,"
#+ an operation that feeds its output back into its input.
# Sometimes the result is a "chaotic" series.
MAX_ITERATIONS=200 #max number of iterations the algorithm can do
# For large seed numbers (>32000), try increasing MAX_ITERATIONS.
h=${1:-$$} # Seed.#Use  PID if user doesnt gice an argument to the scipt
# Use $PID as seed,

#+ if not specified as command-line arg.

echo #Break Line
echo "C($h) -*- $MAX_ITERATIONS Iterations" #Header
echo #Break Line
for ((i=1; i<=MAX_ITERATIONS; i++)) #Loop for maximum number of interations
(200)
do
# echo -n "$h "
# ^^^
# tab
# printf does it better ...

COLWIDTH=%7d #specify column width to wrap function
printf $COLWIDTH $h
let "remainder = h % 2" #Even/Odd checker
if [ "$remainder" -eq 0 ] # Even numbers
then
```

```
let "h /= 2" # Divide by 2.
else #Odd numbers
let "h = h*3 + 1" # Multiply by 3 and add 1.
fi #End if
COLUMNS=10 # Output 10 values per line.
let "line_break = i % $COLUMNS"
if [ "$line_break" -eq 0 ] #check for line break
then
echo #if  == true break line
fi #End if
done #End loop
echo #Line Break
# For info on this math function,
#+ see _Computers, Pattern, Chaos, and Beauty_, by Pickover, p. 185 ff.,
#+ as listed in the bibliography.
exit 0
```

```
jordthanasi@Jords-MacBook-Air ex6 % nano ex6.sh
jordthanasi@Jords-MacBook-Air ex6 % chmod +x ex6.sh
jordthanasi@Jords-MacBook-Air ex6 % ls
ex6.sh
jordthanasi@Jords-MacBook-Air ex6 % ./ex6.sh

C(32828) -*- 200 Iterations

  32828  16414   8207  24622  12311  36934  18467  55402  27701  83104
  41552  20776  10388   5194   2597   7792   3896   1948    974    487
   1462    731   2194   1097   3292   1646    823   2470   1235   3706
   1853   5560   2780   1390    695   2086   1043   3130   1565   4696
   2348   1174    587   1762    881   2644   1322    661   1984    992
    496    248    124     62     31     94     47    142     71    214
    107    322    161    484    242    121    364    182     91    274
    137    412    206    103    310    155    466    233    700    350
    175    526    263    790    395   1186    593   1780    890    445
   1336    668    334    167    502    251    754    377   1132    566
    283    850    425   1276    638    319    958    479   1438    719
   2158   1079   3238   1619   4858   2429   7288   3644   1822    911
   2734   1367   4102   2051   6154   3077   9232   4616   2308   1154
    577   1732    866    433   1300    650    325    976    488    244
    122     61    184     92     46     23     70     35    106     53
    160     80     40     20     10      5     16      8      4      2
      1      4      2      1      4      2      1      4      2      1
      4      2      1      4      2      1      4      2      1      4
      2      1      4      2      1      4      2      1      4      2
      1      4      2      1      4      2      1      4      2      1

jordthanasi@Jords-MacBook-Air ex6 % ./ex6 15
zsh: no such file or directory: ./ex6
jordthanasi@Jords-MacBook-Air ex6 % ./ex6.sh 15

C(15) -*- 200 Iterations

     15     46     23     70     35    106     53    160     80     40
     20     10      5     16      8      4      2      1      4      2
      1      4      2      1      4      2      1      4      2      1
      4      2      1      4      2      1      4      2      1      4
      2      1      4      2      1      4      2      1      4      2
      1      4      2      1      4      2      1      4      2      1
      4      2      1      4      2      1      4      2      1      4
      2      1      4      2      1      4      2      1      4      2
      1      4      2      1      4      2      1      4      2      1
      4      2      1      4      2      1      4      2      1      4
      2      1      4      2      1      4      2      1      4      2
      1      4      2      1      4      2      1      4      2      1
      4      2      1      4      2      1      4      2      1      4
      2      1      4      2      1      4      2      1      4      2
      1      4      2      1      4      2      1      4      2      1
      4      2      1      4      2      1      4      2      1      4
      2      1      4      2      1      4      2      1      4      2
      1      4      2      1      4      2      1      4      2      1
      4      2      1      4      2      1      4      2      1      4
      2      1      4      2      1      4      2      1      4      2

jordthanasi@Jords-MacBook-Air ex6 %
```

Example A-7. *days-between*: Days between two dates

This Bash script days-between.sh calculates the number of days between two dates provided as command-line arguments in MM/DD/YYYY format. It starts by validating that exactly two dates are passed and ensures the dates are after January 3, 1600. It then parses the month, day, and year from each date and uses a simplified version of Gauss's formula to convert each date into a numerical "day index" — a rough count of days since the reference year 1600, accounting for leap years and month shifts. The script computes the difference between these two indices and outputs the absolute number of days between them. It includes safeguards for invalid input, such as incorrect day/month values or improper date formatting. Although effective, the script relies on let, command substitution, and return values for numeric extraction, which could be fragile or imprecise for certain edge cases (e.g., months with fewer than 31 days).

```
#!/bin/bash

# days-between.sh: Number of days between two dates.

# Usage: ./days-between.sh [M]M/[D]D/YYYY [M]M/[D]D/YYYY

ARGS=2 # Two command-line parameters expected.

E_PARAM_ERR=85 # Error code for parameter error.

REFYR=1600 # Reference year for calculations.

CENTURY=100 # Number of years in a century.

DIY=365 # Number of days in a year.

ADJ_DIY=367 # Adjusted days in a year for leap years and fractions.

MIY=12 # Number of months in a year.

DIM=31 # Maximum number of days in a month.

LEAPCYCLE=4 # Leap year cycle.

MAXRETVAL=255 # Largest permissible positive return value from a
function.

# Declare global variables for date difference and absolute value.

diff=

value=

# Declare global variables for day, month, year.

day=

month=

year=

# Function to handle parameter errors.
```

```
Param_Error () {

  echo "Usage: $(basename $0) [M]M/[D]D/YYYY [M]M/[D]D/YYYY"

  echo " (date must be after 1/3/1600)"

  exit $E_PARAM_ERR

}

# Function to parse date from command-line parameters.

Parse_Date () {

  month=${1%%/**} # Extract month.

  dm=${1%/**} # Extract day and month.

  day=${dm#*/} # Extract day.

  year=${1##*/} # Extract year.

}

# Function to check for invalid dates.

check_date () {

  if [ "$day" -gt "$DIM" ] || [ "$month" -gt "$MIY" ] || [ "$year"
-lt "$REFYR" ]; then

    Param_Error # Exit script on invalid date.

  fi

}

# Function to strip leading zeros from day and month.

strip_leading_zero () {

  return ${1#0} # Remove leading zero.

}

# Function to calculate the day index using Gauss' formula.

day_index () {

  day=$1

  month=$2

  year=$3

  let "month = $month - 2" # Adjust month.
```

```
  if [ "$month" -le 0 ]; then
    let "month += 12" # Adjust month for negative values.
    let "year -= 1" # Adjust year.
  fi
  let "year -= $REFYR" # Adjust year by reference year.
  let "indexyr = $year / $CENTURY" # Calculate index year.
  let "Days = $DIY*$year + $year/$LEAPCYCLE - $indexyr +
$indexyr/$LEAPCYCLE + $ADJ_DIY*$month/$MIY + $day - $DIM"
  # Calculate total days using Gauss' formula.
  echo $Days
}
# Function to calculate the difference between two day indices.
calculate_difference () {
  let "diff = $1 - $2" # Calculate difference.
}
# Function to calculate the absolute value.
abs () {
  if [ "$1" -lt 0 ]; then
    let "value = 0 - $1" # Change sign if negative.
  else
    let "value = $1" # Leave it alone if positive.
  fi
}
# Check if the number of command-line parameters is correct.
if [ $# -ne "$ARGS" ]; then
  Param_Error # Call function to handle error.
fi


# Parse the first date.
```

```
Parse_Date $1

check_date $day $month $year # Validate the date.

strip_leading_zero $day # Remove leading zeros from day.

day=$? # Update day variable.

strip_leading_zero $month # Remove leading zeros from month.

month=$? # Update month variable.

let "date1 = $(day_index $day $month $year)" # Calculate day index
for the first date.


# Parse the second date.

Parse_Date $2

check_date $day $month $year # Validate the date.

strip_leading_zero $day # Remove leading zeros from day.

day=$? # Update day variable.

strip_leading_zero $month # Remove leading zeros from month.

month=$? # Update month variable.

date2=$(day_index $day $month $year) # Calculate day index for the
second date.


# Calculate the difference between the two dates.

calculate_difference $date1 $date2

abs $diff # Get the absolute value of the difference.

diff=$value # Update diff variable with the absolute value.


# Output the difference in days.

echo $diff

exit 0 # Exit script successfully.
```

```
jordthanasi@Jords-MacBook-Air ex7 % ls
ex7.sh
jordthanasi@Jords-MacBook-Air ex7 % chmod +x ex7.sh
jordthanasi@Jords-MacBook-Air ex7 % ls
ex7.sh
jordthanasi@Jords-MacBook-Air ex7 % ./ex7.sh 03/29/2020 03/29/2024
1461
jordthanasi@Jords-MacBook-Air ex7 % ./ex7.sh 12/08/2003 10/05/2020
6146
jordthanasi@Jords-MacBook-Air ex7 % ./ex7.sh 01/01/1600 05/05/2020
153526
jordthanasi@Jords-MacBook-Air ex7 % ./ex7.sh 31/12/2020 12/12/2024
Usage: ex7.sh [M]M/[D]D/YYYY [M]M/[D]D/YYYY
 (date must be after 1/3/1600)
jordthanasi@Jords-MacBook-Air ex7 %
```

Example A-8. Making a *dictionary*

This Bash script, makedict.sh, is a utility for generating a clean, sorted dictionary of unique words from one or more text files. It begins by checking if the first provided file is readable; if not, it exits with an error. The core of the script processes the file(s) using a pipeline of text transformation commands: cat reads all the contents, tr A-Z a-z converts all uppercase letters to lowercase, and another tr replaces spaces and non-alphabetic characters with newlines to isolate individual words. It then sorts the words alphabetically using sort, removes duplicates with uniq, and filters out comment lines (those starting with #) and blank lines using grep -v. The result is a clean list of lowercase words—one per line—suitable for use in dictionary files, password cracking tools, or lexicographic analysis. The script exits with the return code of the last command, maintaining good scripting practices.

```
#!/bin/bash

# makedict.sh [make dictionary]

# Modification of /usr/sbin/mkdict (/usr/sbin/cracklib-forman) script.

# Original script copyright 1993, by Alec Muffett.

#

# This modified script included in this document in a manner

#+ consistent with the "LICENSE" document of the "Crack" package

#+ that the original script is a part of.

# This script processes text files to produce a sorted list

#+ of words found in the files.

# This may be useful for compiling dictionaries

#+ and for other lexicographic purposes.


E_BADARGS=85 # Error code for bad arguments.


# Check if at least one valid file argument is provided and readable.

if [ ! -r "$1" ]; then

  echo "Usage: $0 files-to-process"

  exit $E_BADARGS # Exit with error code if no valid file is provided.

fi
```

```
# Process the specified files.

cat $* |       # Dump the contents of all specified files to stdout.

tr A-Z a-z |   # Convert all uppercase letters to lowercase.

tr ' ' '\012' |# Convert spaces to newlines to ensure each word is on a
new line.

# tr -cd '\012[a-z][0-9]' | # Original script: Remove all
non-alphanumeric characters.

tr -c '\012a-z' '\012' | # Instead of deleting non-alpha characters,
convert them to newlines.

sort |         # Sort the words alphabetically.

uniq |         # Remove duplicate words.

grep -v '^#' | # Remove lines that start with a hash (#).

grep -v '^$'   # Remove blank lines.



exit $? # Exit with the status of the last command.
```

```
jordthanasi@Jords-MacBook-Air ex8 % nano ex8.sh
jordthanasi@Jords-MacBook-Air ex8 % chmod +x ex8.sh
jordthanasi@Jords-MacBook-Air ex8 % ls
ex8.sh
jordthanasi@Jords-MacBook-Air ex8 % touch file1.txt file2.txt
jordthanasi@Jords-MacBook-Air ex8 % ls
ex8.sh          file1.txt       file2.txt
jordthanasi@Jords-MacBook-Air ex8 % cat file1.txt
This is a test text composed of some phrases.%
jordthanasi@Jords-MacBook-Air ex8 % cat file2.txt
This is also a test text composed of different phrases.%
jordthanasi@Jords-MacBook-Air ex8 % ./ex8.sh file1.txt file2.txt
a
also
composed
different
is
of
phrases
some
test
text
this
jordthanasi@Jords-MacBook-Air ex8 %
```

Example A-9. Soundex conversion

This Bash script, soundex.sh, calculates the Soundex code for a given name — a phonetic encoding system that helps group similar-sounding names. It requires exactly one command-line argument (the name). The script begins by converting the name to lowercase, extracting and capitalizing the first letter (which becomes the prefix of the Soundex code), then mapping the remaining letters to digits based on their phonetic groupings (e.g., b/f/p/v = 1, c/g/j/k = 2, etc.). Vowels and the letters w and h are removed, and repeated digits are compressed using tr -s. A special "Exception Patch" section attempts to ensure that the numeric encoding properly handles cases where the first character is a silent or vowel-like sound. After processing, the script pads the numeric suffix with zeros and concatenates it with the prefix, truncating the final result to exactly four characters. This approach mimics how the Soundex algorithm is used by U.S. government agencies and genealogists to match names with similar pronunciation but different spellings (e.g., "Smith" and "Smythe" both become S530).

```
#!/bin/bash

# soundex.sh: Calculate "soundex" code for names

# ======================================================

# Soundex script

# by

# Mendel Cooper

# thegrendel.abs@gmail.com

# reldate: 23 January, 2002

#

# Placed in the Public Domain.

#

# A slightly different version of this script appeared in

# Ed Schaefer's July, 2002 "Shell Corner" column

# in "Unix Review" on-line,

# http://www.unixreview.com/documents/uni1026336632258/

# ======================================================

ARGCOUNT=1  # Expect exactly one argument.

E_WRONGARGS=90  # Error code for incorrect number of arguments.


# Check if the number of arguments is correct.
```

```
if [ $# -ne "$ARGCOUNT" ]; then
  echo "Usage: $(basename $0) name"  # Print usage message.
  exit $E_WRONGARGS  # Exit with error code.
fi

# Function to assign numerical values to letters of the name.
assign_value () {
  val1=bfpv  # 'b,f,p,v' = 1
  val2=cgjkqsxz  # 'c,g,j,k,q,s,x,z' = 2
  val3=dt  # 'd,t' = 3
  val4=l  # 'l' = 4
  val5=mn  # 'm,n' = 5
  val6=r  # 'r' = 6

  # Assign numerical values, remove duplicates, and strip vowels.
  value=$(echo "$1" | tr -d wh \
    | tr $val1 1 | tr $val2 2 | tr $val3 3 \
    | tr $val4 4 | tr $val5 5 | tr $val6 6 \
    | tr -s 123456 \
    | tr -d aeiouy)
}

input_name="$1"  # Store input name.
echo
echo "Name = $input_name"

# Change all characters of name input to lowercase.
name=$(echo $input_name | tr A-Z a-z)
```

```
# Prefix of soundex code: first letter of name.
char_pos=0  # Initialize character position.
prefix0=${name:$char_pos:1}  # Get first letter.
prefix=$(echo $prefix0 | tr a-z A-Z)  # Convert to uppercase.

let "char_pos += 1"  # Move to the second letter.
name1=${name:$char_pos}  # Get the rest of the name.

# Exception Patch to handle special cases.
char1=$(echo $prefix | tr A-Z a-z)  # First letter in lowercase.
assign_value $name  # Assign values to the name.
s1=$value
assign_value $name1  # Assign values to the name shifted one character.
s2=$value

assign_value $char1  # Assign value to the first letter.
s3=$value
s3=9$s3  # If the first letter is a vowel, 'w', or 'h', set value to '9'.

# Determine the suffix based on special conditions.
if [[ "$s1" -ne "$s2" || "$s3" -eq 9 ]]; then
  suffix=$s2
else
  suffix=${s2:$char_pos}
fi

padding=000  # Use at most 3 zeros to pad.
```

```
soun=$prefix$suffix$padding  # Combine prefix, suffix, and padding.
MAXLEN=4  # Maximum length of soundex code.
soundex=${soun:0:$MAXLEN}  # Truncate to 4 characters.
echo "Soundex = $soundex"
echo

# The soundex code is a method of indexing and classifying names
# by grouping together the ones that sound alike.
# The soundex code for a given name is the first letter of the name,
# followed by a calculated three-number code.
# Similar sounding names should have almost the same soundex codes.
# Examples:
# Smith and Smythe both have a "S-530" soundex.
# Harrison = H-625
# Hargison = H-622
# Harriman = H-655
# This works out fairly well in practice, but there are numerous anomalies.
#
# The U.S. Census and certain other governmental agencies use soundex,
# as do genealogical researchers.
# For more information,
# see the "National Archives and Records Administration home page",
# http://www.nara.gov/genealogy/soundex/soundex.html
# Exercise:
# --------
# Simplify the "Exception Patch" section of this script.

exit 0  # Exit the script.
```

Simplified Exeption Patch:

```
# Assign soundex values (excluding first letter).
assign_value "$name1"
suffix=$value

# Remove leading digit if it matches encoded first letter.
assign_value "$prefix0"
first_digit=$value

# If the suffix starts with the same digit as the first letter's soundex value, drop it.
[[ "${suffix:0:1}" == "$first_digit" ]] && suffix=${suffix:1}
```

```
jordthanasi@Jords-MacBook-Air ex9 % nano ex9.sh
jordthanasi@Jords-MacBook-Air ex9 % chmod +x ex9.sh
jordthanasi@Jords-MacBook-Air ex9 % ls
ex9.sh
jordthanasi@Jords-MacBook-Air ex9 % ./ex9.sh Jord

Name = Jord
Soundex = J630

jordthanasi@Jords-MacBook-Air ex9 % ./ex9.sh Enkel

Name = Enkel
Soundex = E524

jordthanasi@Jords-MacBook-Air ex9 % ./ex9.sh Filani

Name = Filani
Soundex = F450

jordthanasi@Jords-MacBook-Air ex9 % ./ex9.sh Fisteku

Name = Fisteku
Soundex = F232

jordthanasi@Jords-MacBook-Air ex9 %
```

Example A-10. *Game of Life*

This Bash script, life.sh, simulates Conway's Game of Life—a cellular automaton where each cell in a grid is either alive (.) or dead (_) and evolves over discrete generations based on simple rules: a living cell with 2 or 3 neighbors survives, a dead cell with exactly 3 neighbors becomes alive, and all others die or remain dead. The script reads an initial configuration from a file (gen0), processes the grid as a 1D array, and applies neighbor-counting logic to determine each cell's state in the next generation, displaying the grid with a 2-second delay between steps. It handles edge boundaries by treating them as always dead and exits early if no cells remain alive. While functional for fixed-size grids, the script includes exercises to enhance it—such as enabling grid wrapping, detecting grid size from the input file, and optimizing redundant code—making it both a working demo and a learning tool for Bash scripting.

Exercise: Simplify the script, it has redundant code

```
#!/bin/bash

# life.sh: "Life in the Slow Lane"

# Author: Mendel Cooper

# License: GPL3

# Version 0.2: Patched by Daniel Albers to allow non-square grids as input.

# Version 0.2.1: Added 2-second delay between generations.

# This is the Bash script version of John Conway's "Game of Life".

# "Life" is a simple implementation of cellular automata.


# On a rectangular grid, let each "cell" be either "living" or "dead."

# Designate a living cell with a dot, and a dead one with a blank space.

# Begin with an arbitrarily drawn dot-and-blank grid, and let this be

# the starting generation: generation 0.

# Determine each successive generation by the following rules:

# 1) Each cell has 8 neighbors: left, right, top, bottom, and the 4 diagonals.

# 2) A living cell with either 2 or 3 living neighbors remains alive.

SURVIVE=2

# 3) A dead cell with 3 living neighbors comes alive, a "birth."

BIRTH=3

# 4) All other cases result in a dead cell for the next generation.


# Read the starting generation from the file "gen0" or specified file.

startfile=gen0

if [ -n "$1" ]; then

  startfile="$1"

fi


# Abort script if "startfile" not specified and default file "gen0" not
present.
```

```
E_NOSTARTFILE=86
if [ ! -e "$startfile" ]; then
  echo "Startfile \"$startfile\" missing!"
  exit $E_NOSTARTFILE
fi


ALIVE1=.
DEAD1=_
# Represent living and dead cells in the start-up file.


# Grid size (can be changed to match desired grid size).
ROWS=10
COLS=10


GENERATIONS=10 # Number of generations to cycle through.
NONE_ALIVE=85 # Exit status if no cells left alive.
DELAY=2 # Pause between generations.


TRUE=0
FALSE=1
ALIVE=0
DEAD=1
avar= # Global; holds current generation.
generation=0 # Initialize generation count.


# Calculate the number of cells.
let "cells = $ROWS * $COLS"
```

```
# Arrays containing "cells."

declare -a initial

declare -a current


# Function to display the grid.

display () {

  alive=0 # How many cells alive at any given time.

  declare -a arr

  arr=( `echo "$1"` ) # Convert passed arg to array.

  element_count=${#arr[*]}

  local i

  local rowcheck


  for ((i=0; i<$element_count; i++)); do

    # Insert newline at end of each row.

    let "rowcheck = $i % COLS"

    if [ "$rowcheck" -eq 0 ]; then

      echo # Newline.

      echo -n " " # Indent.

    fi

    cell=${arr[i]}

    if [ "$cell" = . ]; then

      let "alive += 1"

    fi

    echo -n "$cell" | sed -e 's/_/ /g' # Print out array, changing underscores
to spaces.

  done

  return

}
```

```
# Function to test if cell coordinate is valid.
IsValid () {
  if [ -z "$1" -o -z "$2" ]; then # Mandatory arguments missing?
    return $FALSE
  fi
  local row
  local lower_limit=0 # Disallow negative coordinate.
  local upper_limit
  local left
  local right
  let "upper_limit = $ROWS * $COLS - 1" # Total number of cells.
  if [ "$1" -lt "$lower_limit" -o "$1" -gt "$upper_limit" ]; then
    return $FALSE # Out of array bounds.
  fi
  row=$2
  let "left = $row * $COLS" # Left limit.
  let "right = $left + $COLS - 1" # Right limit.
  if [ "$1" -lt "$left" -o "$1" -gt "$right" ]; then
    return $FALSE # Beyond row boundary.
  fi
  return $TRUE # Valid coordinate.
}

# Function to test whether cell is alive.
IsAlive () {
  GetCount "$1" $2 # Get alive cell count in neighborhood.
  local nhbd=$?
```

```
  if [ "$nhbd" -eq "$BIRTH" ]; then # Alive in any case.
    return $ALIVE
  fi
  if [ "$3" = "." -a "$nhbd" -eq "$SURVIVE" ]; then # Alive only if previously
alive.
    return $ALIVE
  fi
  return $DEAD # Defaults to dead.
}


# Function to count live cells in passed cell's neighborhood.
GetCount () {
  local cell_number=$2
  local array
  local top
  local center
  local bottom
  local r
  local row
  local i
  local t_top
  local t_cen
  local t_bot
  local count=0
  local ROW_NHBD=3
  array=( `echo "$1"` )
  let "top = $cell_number - $COLS - 1" # Set up cell neighborhood.
  let "center = $cell_number - 1"
  let "bottom = $cell_number + $COLS - 1"
```

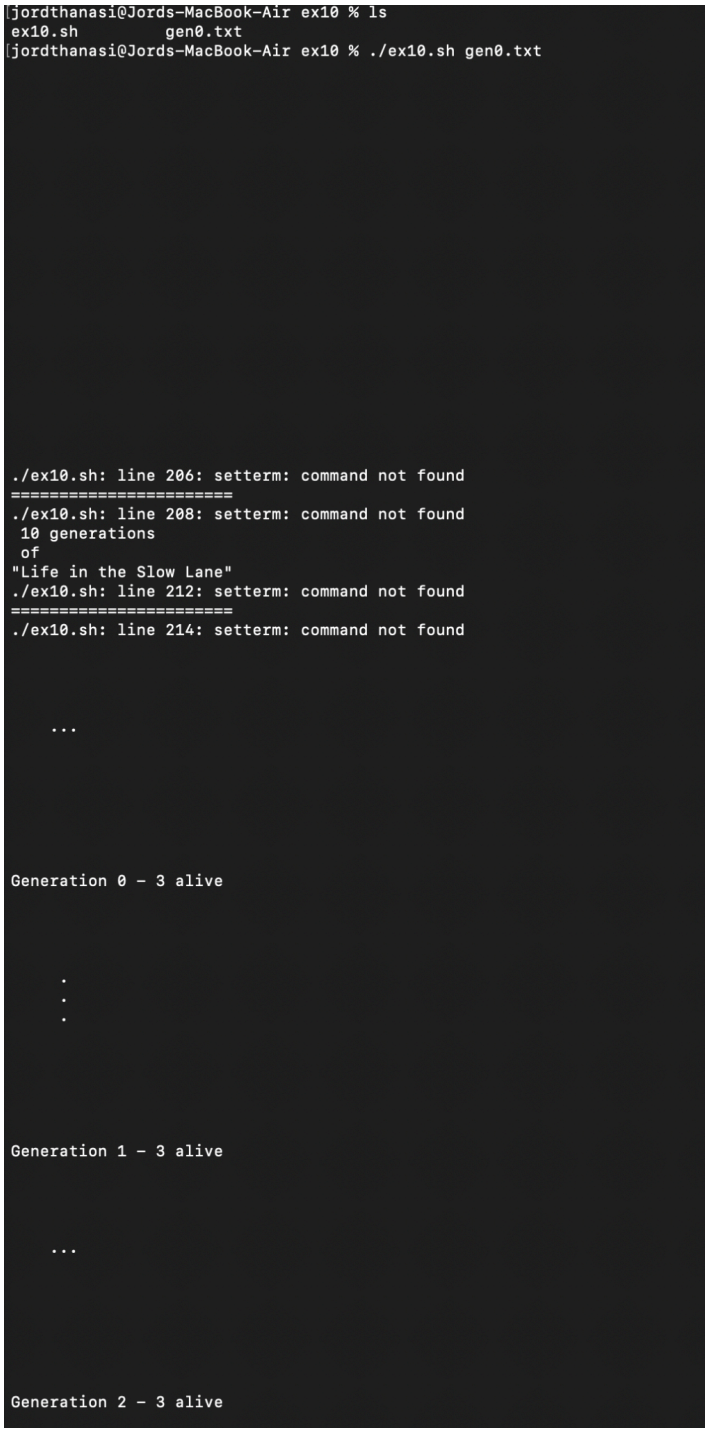
```
let "r = $cell_number / $COLS"
for ((i=0; i<$ROW_NHBD; i++)); do # Traverse from left to right.
  let "t_top = $top + $i"
  let "t_cen = $center + $i"
  let "t_bot = $bottom + $i"
  let "row = $r" # Count center row.
  IsValid $t_cen $row # Valid cell position?
  if [ $? -eq "$TRUE" ]; then
    if [ ${array[$t_cen]} = "$ALIVE1" ]; then # Is it alive?
      let "count += 1" # Increment count.
    fi
  fi
  let "row = $r - 1" # Count top row.
  IsValid $t_top $row
  if [ $? -eq "$TRUE" ]; then
    if [ ${array[$t_top]} = "$ALIVE1" ]; then # Redundancy here.
      let "count += 1"
    fi
  fi
  let "row = $r + 1" # Count bottom row.
  IsValid $t_bot $row
  if [ $? -eq "$TRUE" ]; then
    if [ ${array[$t_bot]} = "$ALIVE1" ]; then
      let "count += 1"
    fi
  fi
done
if [ ${array[$cell_number]} = "$ALIVE1" ]; then
```

```
    let "count -= 1" # Make sure value of tested cell itself is not counted.
  fi
  return $count
}


# Function to update generation array.
next_gen () {
  local array
  local i=0
  array=( `echo "$1"` ) # Convert passed arg to array.


  while [ "$i" -lt "$cells" ]; do
    IsAlive "$1" $i ${array[$i]} # Is the cell alive?
    if [ $? -eq "$ALIVE" ]; then # If alive, then
      array[$i]=. # represent the cell as a period.
    else
      array[$i]="_" # Otherwise underscore (will later be converted to space).
    fi
    let "i += 1"
  done


  avar=`echo ${array[@]}` # Convert array back to string variable.
  display "$avar" # Display it.
  echo; echo
  echo "Generation $generation - $alive alive"
  if [ "$alive" -eq 0 ]; then
    echo
    echo "Premature exit: no more cells alive!"
```

```
    exit $NONE_ALIVE # No point in continuing if no live cells.

  fi

}



# Main

initial=( `cat "$startfile" | sed -e '/#/d' | tr -d '\n' | sed -e 's/\./\. /g'
-e 's/_/_ /g'` )

# Load initial array with contents of startup file, removing comments and
formatting.

clear # Clear screen.

echo # Title

setterm -reverse on

echo "========================"

setterm -reverse off

echo " $GENERATIONS generations"

echo " of"

echo "\"Life in the Slow Lane\""

setterm -reverse on

echo "========================"

setterm -reverse off



sleep $DELAY # Display "splash screen" for 2 seconds.



# Display first generation.

Gen0=`echo ${initial[@]}`

display "$Gen0" # Display only.

echo; echo

echo "Generation $generation - $alive alive"

sleep $DELAY
```

```
let "generation += 1" # Bump generation count.
echo

# Display second generation.
Cur=`echo ${initial[@]}`
next_gen "$Cur" # Update & display.
sleep $DELAY
let "generation += 1" # Increment generation count.

# Main loop for displaying subsequent generations.
while [ "$generation" -le "$GENERATIONS" ]; do
  Cur="$avar"
  next_gen "$Cur"
  let "generation += 1"
  sleep $DELAY
done

exit 0 # End of script.

# The grid in this script has a "boundary problem."
# The top, bottom, and sides border on a void of dead cells.
# Exercise: Change the script to have the grid wrap around,
# so that the left and right sides will "touch," as will the top and bottom.

# Exercise: Create a new "gen0" file to seed this script.
# Use a 12 x 16 grid, instead of the original 10 x 10 one.
# Make the necessary changes to the script so it will run with the altered
file.
```

```
# Exercise: Modify this script so that it can determine the grid size

# from the "gen0" file, and set any variables necessary for the script to run.

# This would make unnecessary any changes to variables in the script for an
altered grid size.


# Exercise: Optimize this script. It has redundant code.
```



```
Generation 2 — 3 alive


     .
     .
     .




Generation 3 — 3 alive


    ...




Generation 4 — 3 alive


     .
     .
     .




Generation 5 — 3 alive


    ...




Generation 6 — 3 alive


     .
     .
     .




Generation 7 — 3 alive


    ...
```

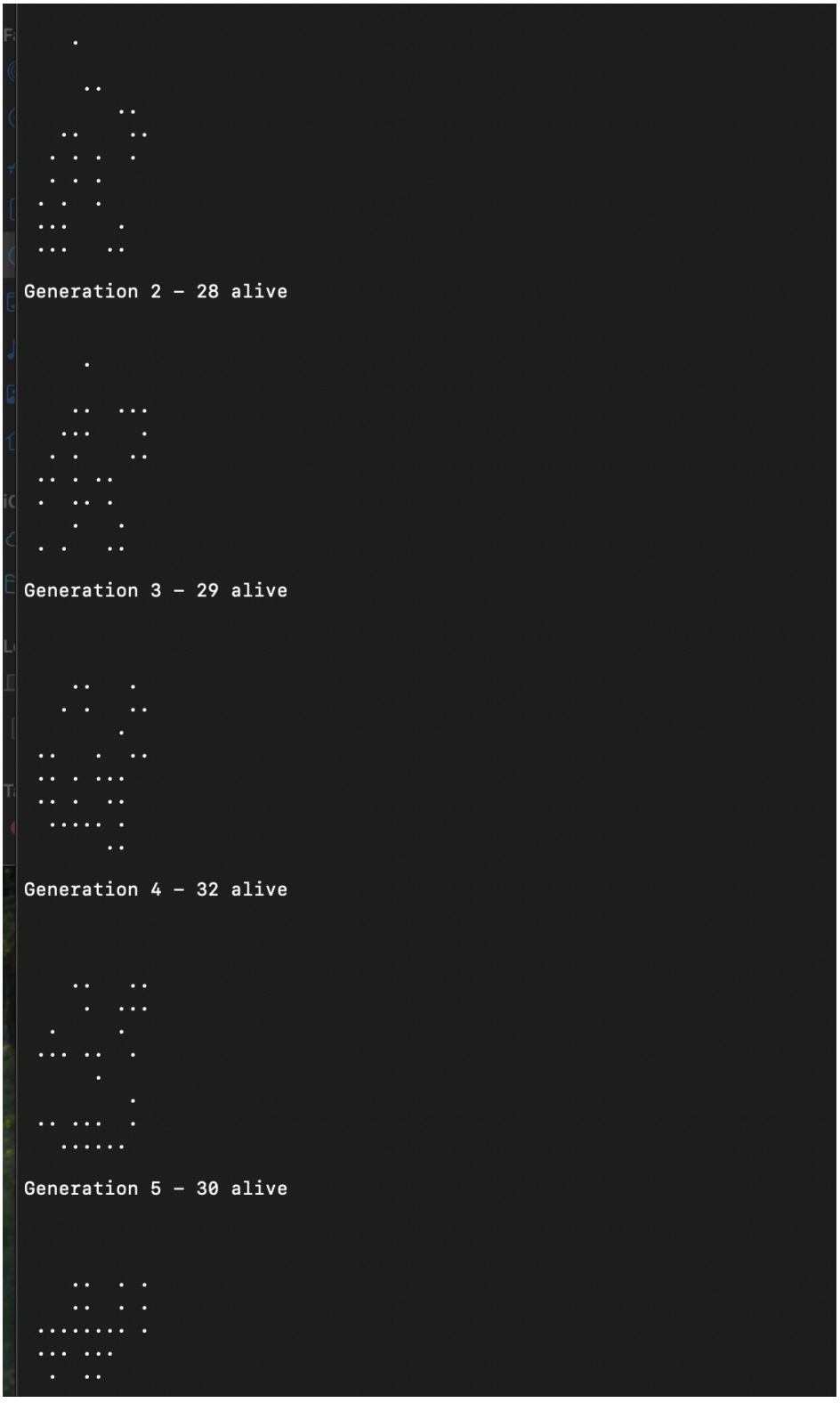
```
    ...




Generation 6 - 3 alive


     .
     .
     .




Generation 7 - 3 alive


    ...




Generation 8 - 3 alive


     .
     .
     .




Generation 9 - 3 alive


    ...




Generation 10 - 3 alive
jordthanasi@Jords-MacBook-Air ex10 % cat gen0.txt
----------
----------
----------
---...----
----------
----------
----------
----------
----------
----------%
jordthanasi@Jords-MacBook-Air ex10 %
```

Example A-11. Data file for *Game of Life*

```
# gen0
#
# This is an example "generation 0" start-up file for "life.sh". #
---------------------------------------------------------------- # The
"gen0" file is a 10 x 10 grid using a period (.) for live cells, #+ and
an underscore (_) for dead ones. We cannot simply use spaces #+ for dead
cells in this file because of a peculiarity in Bash arrays. # [Exercise
for the reader: explain this.]
#
# Lines beginning with a '#' are comments, and the script ignores them.
__.__..___
__..._.____
____.___..
_.________.
____._____
 ..__...___
____._____
___...____
__..._..___
_..___..__
```

The following script is by Mark Moraes of the University of Toronto. See the file `Moraes-COPYRIGHT` for permissions and restrictions. This file is included in the combined HTML/source tarball of the *ABS Guide*.

```
jordthanasi@Jords-MacBook-Air ex10 % touch gen1.txt
jordthanasi@Jords-MacBook-Air ex10 % cat gen1.txt
__._._.__.__
__.._____._
___._____.
__.._____
__.._____.._
_.._____._
_____.._____
_._._._____
_._____.._
_.._____.._%
jordthanasi@Jords-MacBook-Air ex10 % ./ex10.sh gen1.txt
./ex10.sh: line 206: setterm: command not found
=======================
./ex10.sh: line 208: setterm: command not found
 10 generations
 of
"Life in the Slow Lane"
./ex10.sh: line 212: setterm: command not found
=======================
./ex10.sh: line 214: setterm: command not found

    . .  .
    ..    .
     .     .
    ..
    ..   ..
   ..     .
       ..
   .   . .
   .     ..
   ..  ..

Generation 0 - 29 alive


    .
    . .   .
      .
      .   .
         ..
   ...   . .
   ..  ...
        .
  ..     .
   ..   ..

Generation 1 - 27 alive
```


Generation 2 — 28 alive
Generation 3 — 29 alive
Generation 4 — 32 alive
Generation 5 — 30 alive


Generation 5 – 30 alive
Generation 6 – 33 alive
Generation 7 – 21 alive
Generation 8 – 23 alive
Generation 9 – 14 alive


Generation 6 – 33 alive
Generation 7 – 21 alive
Generation 8 – 23 alive
Generation 9 – 14 alive
Generation 10 – 12 alive
jordthanasi@Jords-MacBook-Air ex10 %

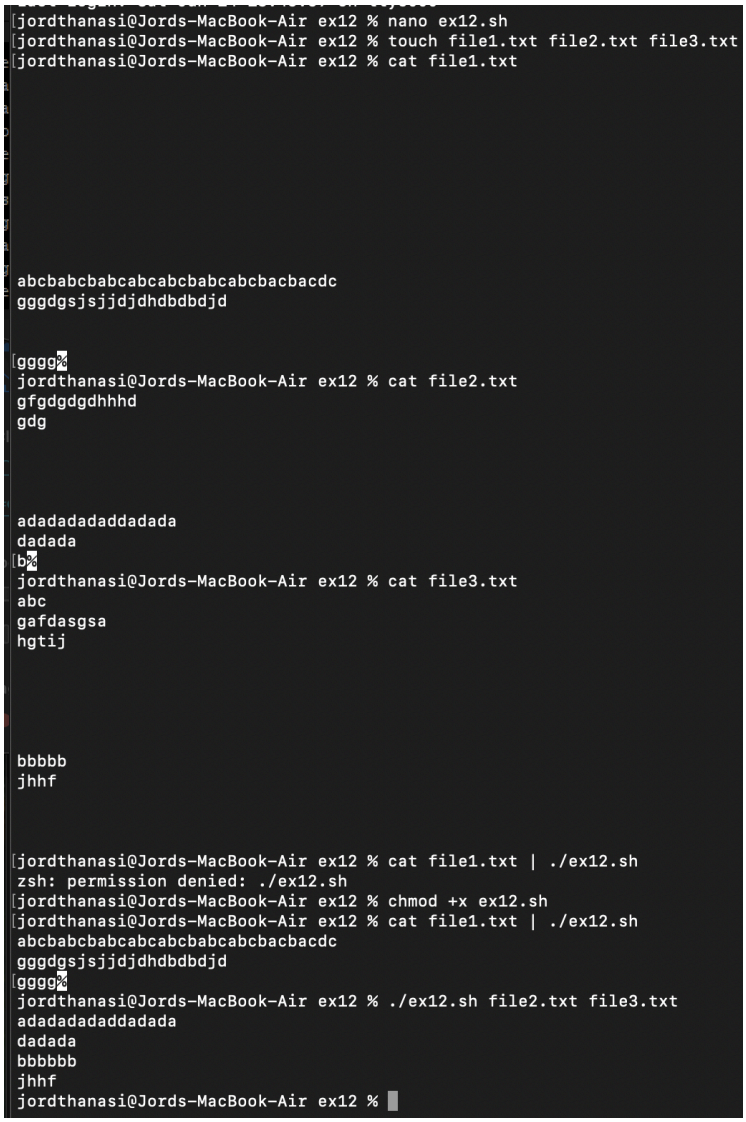
Example A-12. *behead*: Removing mail and news message headers

This shell script is designed to strip the header from email or Usenet message files, which typically ends at the first empty line. If no arguments are provided, the script assumes the input comes from standard input (e.g., piped or redirected file) and uses sed to remove all lines from the start of the file up to and including the first empty line (1,/^$/d), as well as any lines that are completely blank or only contain spaces (/^[ ]*$/d). If filenames are provided as arguments, the script loops through each file and applies the same sed logic to clean each one. It ends with a note suggesting enhancements such as adding error checking (e.g., for missing files) and possibly embedding the repeated sed command in a function, which could improve maintainability—though in a small script like this, a function might not be strictly necessary. The script is open source, released under a permissive license by the University of Toronto, with attribution and disclaimer clauses.

```
#!/bin/sh
# Strips off the header from a mail/News message i.e. till the first empty
line.
# Author: Mark Moraes, University of Toronto
# ==> These comments added by author of this document.

# Check if no command-line arguments are present.
if [ $# -eq 0 ]; then
  # ==> If no command-line args present, then work on file redirected to
stdin.
  sed -e '1,/^$/d' -e '/^[ ]*$/d'
  # --> Delete all lines until the first empty line (header) and then delete
empty lines.
else
  # ==> If command-line args present, then work on files named.
  for i; do
    sed -e '1,/^$/d' -e '/^[ ]*$/d' $i
    # --> For each file, delete all lines until the first empty line and then
delete empty lines.
  done
fi

exit

# ==> Exercise: Add error checking and other options.
# ==> Note that the small sed script repeats, except for the arg passed.
# ==> Does it make sense to embed it in a function? Why or why not?

/*
* Copyright University of Toronto 1988, 1989.
* Written by Mark Moraes
*
* Permission is granted to anyone to use this software for any purpose on
* any computer system, and to alter it and redistribute it freely, subject
* to the following restrictions:
*
* 1. The author and the University of Toronto are not responsible
```

```
 *    for the consequences of use of this software, no matter how awful,
 *    even if they arise from flaws in it.
 *
 * 2. The origin of this software must not be misrepresented, either by
 *    explicit claim or by omission. Since few users ever read sources,
 *    credits must appear in the documentation.
 *
 * 3. Altered versions must be plainly marked as such, and must not be
 *    misrepresented as being the original software. Since few users
 *    ever read sources, credits must appear in the documentation.
 *
 * 4. This notice may not be removed or altered.
 */
```

+Antek Sawicki contributed the following script, which makes very clever use of the parameter substitution operators discussed in Section 10.2.



Yes, embedding the repeated sed command in a function makes sense because it improves the script's clarity, avoids code duplication, and makes future maintenance easier. By placing the sed logic into a function (e.g., strip_header), the script becomes more readable and changes to the stripping logic only need to be made in one place. While functions may seem unnecessary in very short scripts, using one here is justified since the sed command is repeated and may be extended with error handling or logging later. It also aligns with good scripting practices for modularity and scalability.

Example A-13. *password*: Generating random 8-character passwords

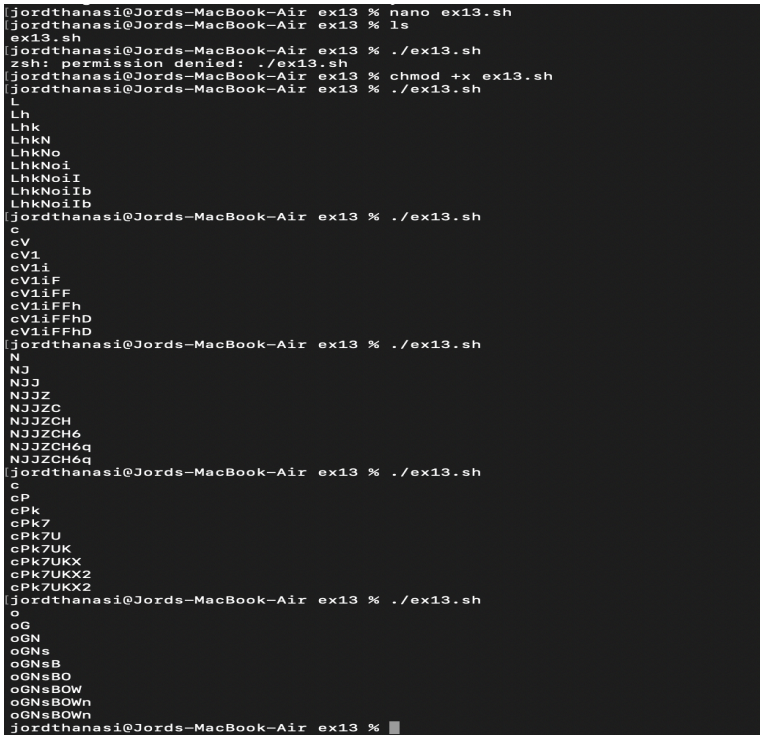
This Bash script generates a random alphanumeric password by selecting characters from a predefined set (MATRIX) consisting of digits, uppercase, and lowercase letters. The desired password length is set by the LENGTH variable (default is 8 characters). The script uses a while loop that initializes a counter n (defaulting to 1 if unset) and runs until the counter exceeds LENGTH. In each iteration, it appends a randomly selected character from MATRIX to the PASS variable using ${MATRIX:$(($RANDOM%${#MATRIX})):1}—a syntax that selects a single character at a random index within the character set. The password is built one character at a time and optionally displayed during construction (the echo "$PASS" line). After the loop, the final password is printed, and the script exits normally with exit 0. This script demonstrates simple but effective password generation using core Bash features without external dependencies.

```
#!/bin/bash

#

# This script generates a random password consisting of

alphanumeric characters.

# Author: Antek Sawicki <tenox@tenox.tc>

# Usage permission granted to the ABS Guide author.

#



# Define the character set from which the password will be

generated.

MATRIX="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmno

pqrstuvwxyz"



# Define the length of the password.
```

```
LENGTH="8" # Change this value to generate passwords of
different lengths.



# Start a loop to generate the password.
while [ "${n:=1}" -le "$LENGTH" ]
# If 'n' is not initialized, set it to 1.
do
    # Generate a random character from the MATRIX and
append it to the password.
    PASS="$PASS${MATRIX:$(($RANDOM%${#MATRIX})):1}" #Each
iteration appends a character of the matrix until n==LENGTH
    # The ${#MATRIX} gets the length of the MATRIX.
    # $RANDOM generates a random number, and %${#MATRIX}
ensures it's within the range of the MATRIX length.
    # ${MATRIX:$(($RANDOM%${#MATRIX})):1} gets a character
from the MATRIX at a random position.



    # Uncomment the line below to see the password being
built character by character.
     echo "$PASS"
```

```
    # Increment 'n' for the next iteration.

    let n+=1

done



# Output the generated password.

echo "$PASS"



# Exit the script with status 0.

exit 0
```

James R. Van Zandt contributed this script which uses named pipes and, in his words, "really exercises quoting and escaping."



Example A-14. *fifo*: Making daily backups, using named pipes

This script performs a secure, compressed remote backup of essential system directories using a named pipe (/pipe). It begins by defining the local host's name (HERE) and the remote host (THERE), then prints a timestamped message. It ensures that /pipe is a valid named pipe by deleting any existing file with that name and recreating it using mkfifo. A background process (su xyz -c ...) switches to user xyz and uses ssh to connect to the remote host bilbo, where it saves the incoming data to a compressed archive file. Meanwhile, on the local machine, tar compresses system directories and writes the output to /pipe, which is read in real-time by the remote ssh command. The use of a named pipe allows two independent processes (tar and ssh) to communicate asynchronously through the filesystem. This wouldn't work with a standard anonymous pipe (|) because the writer and reader would need to be in the same shell pipeline; in this case, they are launched as separate processes. Finally, while the script does not delete the named pipe after use, doing so is recommended (e.g., rm -f /pipe) to avoid leaving unused filesystem entries.

```
#!/bin/bash

# ==> Script by James R. Van Zandt, and used here with his permission.

# ==> Comments added by author of this document.



HERE=`uname -n` # ==> hostname

THERE=bilbo

echo "starting remote backup to $THERE at `date +%r`"

# ==> `date +%r` returns time in 12-hour format, i.e. "08:08:34 PM".

# make sure /pipe really is a pipe and not a plain file

rm -rf /pipe

mkfifo /pipe # ==> Create a "named pipe", named "/pipe" ...
```

```
# ==> 'su xyz' runs commands as user "xyz".

# ==> 'ssh' invokes secure shell (remote login client).

su xyz -c "ssh $THERE \"cat > /home/xyz/backup/${HERE}-daily.tar.gz\" <

/pipe"&

cd /

tar -czf - bin boot dev etc home info lib man root sbin share usr var >

/pipe

# ==> Uses named pipe, /pipe, to communicate between processes:

# ==> 'tar/gzip' writes to /pipe and 'ssh' reads from /pipe.

# ==> The end result is this backs up the main directories, from / on down.

# ==> What are the advantages of a "named pipe" in this situation,

# ==>+ as opposed to an "anonymous pipe", with |?

# ==> Will an anonymous pipe even work here?

# ==> Is it necessary to delete the pipe before exiting the script?

# ==> How could that be done?

exit 0
```

+

Stéphane Chazelas used the following script to demonstrate generating prime numbers without arrays.

```
[jordthanasi@Jords-MacBook-Air ex14 % ls
ex14.sh
[jordthanasi@Jords-MacBook-Air ex14 % chmod +x ex14.sh
[jordthanasi@Jords-MacBook-Air ex14 % ./ex14.sh
starting remote backup to bilbo at 02:13:01 PM
mkfifo: /pipe: Read-only file system
./ex14.sh: line 16: syntax error near unexpected token `newline'
./ex14.sh: line 16: `tar -czf - bin boot dev etc home info lib man root sbin share usr var >'
jordthanasi@Jords-MacBook-Air ex14 % ./ex14.sh: line 14: $PIPE: ambiguous redirect
[jordthanasi@Jords-MacBook-Air ex14 % ./ex14.sh
starting remote backup to bilbo at 02:13:54 PM
mkfifo: /pipe: Read-only file system
./ex14.sh: line 17: syntax error near unexpected token `newline'
./ex14.sh: line 17: `tar -czf - bin boot dev etc home info lib man root sbin share usr var >'
jordthanasi@Jords-MacBook-Air ex14 % su: Sorry
```

Example A-15. Generating prime numbers using the modulo operator

This Bash script (primes.sh) generates prime numbers from 2 up to a defined limit (1000) using a recursive function rather than traditional loops or arrays. It starts with an initial call Primes 1, which then recursively increments the number and checks whether it is divisible by any previously found primes passed as parameters. If a number has no divisors (i.e., it is prime), it is added to the argument list and passed to the next recursive call. The check ((i * i > n)) is an optimization that stops the inner loop early once all relevant divisors are tested. The use of "$@" allows the function to accumulate and pass along all found primes without using arrays. When the counter reaches the LIMIT, the function prints the list of prime numbers and exits. This intuitive method mimics manual prime checking and cleverly uses positional parameters and recursion to track state, making it a compact and elegant demonstration of functional-style logic in Bash.

```
#!/bin/bash
# primes.sh: Generate prime numbers, without using arrays.
# Script contributed by Stephane Chazelas.

# This does *not* use the classic "Sieve of Eratosthenes" algorithm, #+
but instead the more intuitive method of testing each candidate number
#+ for factors (divisors), using the "%" modulo operator.


LIMIT=1000 # Primes, 2 ... 1000.

Primes()
 {
  (( n = $1 + 1 )) # Bump to next integer.
 shift # Next parameter in list.
```

```
# echo "_n=$n i=$i_"

 if (( n == LIMIT ))
 then echo $*
 return
 fi

 for i; do # "i" set to "@", previous values of $n. # echo "-n=$n
i=$i-"
 (( i * i > n )) && break # Optimization.
 (( n % i )) && continue # Sift out non-primes using modulo operator.
Primes $n $@ # Recursion inside loop.
 return
 done

 Primes $n $@ $n # Recursion outside loop.
 # Successively accumulate
                           #+ positional parameters.
 # "$@" is the accumulating list of primes. }

Primes 1

exit $?

# Pipe output of the script to 'fmt' for prettier printing. #

Uncomment lines 16 and 24 to help figure out what is going on.

# Compare the speed of this algorithm for generating primes
#+ with the Sieve of Eratosthenes (ex68.sh).


# Exercise: Rewrite this script without recursion.
```

Solution:

```
#!/bin/bash
# primes.sh: Generate prime numbers, without using arrays.
# Script contributed by Stephane Chazelas.
# This does *not* use the classic "Sieve of Eratosthenes" algorithm,
# but instead the more intuitive method of testing each candidate number
# for factors (divisors), using the "%" modulo operator.

LIMIT=1000 # Primes, 2 ... 1000.
# Set the limit up to which we want to find prime numbers.

Primes()
{
   (( n = $1 + 1 )) # Bump to next integer.
   # Increment the input number by 1.
```

```
  shift # Next parameter in list.
  # Shift the positional parameters to the left, discarding the first one.

  # echo "_n=$n i=$i_"
  # Uncomment for debugging: shows current value of n and i.

  if (( n == LIMIT ))
  then
    echo $*
    # If the current number equals the limit, print all the primes found so far.
    return
  fi

  for i; do # "i" set to "@", previous values of $n.
    # Iterate over all passed parameters, which are potential primes.

    # echo "-n=$n i=$i-"
    # Uncomment for debugging: shows current value of n and i.

    (( i * i > n )) && break # Optimization.
    # If the square of i is greater than n, exit the loop.

    (( n % i )) && continue # Sift out non-primes using modulo operator.
    # If n is not divisible by i, continue to the next iteration.

    Primes $n $@ # Recursion inside loop.
    # Recursively call Primes with the current number and all primes found so far.

    return
  done

  Primes $n $@ $n # Recursion outside loop.
  # Recursively call Primes with the current number, all primes found so far, and the current
number itself.

  # "$@" is the accumulating list of primes.
}

Primes 1
# Initial call to the Primes function with 1.

exit $?
# Exit the script with the status of the last command.
```

```
jordthanasi@Jords-MacBook-Air ex15 % nano ex15.sh
jordthanasi@Jords-MacBook-Air ex15 % ls
ex15.sh
jordthanasi@Jords-MacBook-Air ex15 % ./ex15.sh
zsh: permission denied: ./ex15.sh
jordthanasi@Jords-MacBook-Air ex15 % chmod +x ex15.sh
jordthanasi@Jords-MacBook-Air ex15 % ./ex15.sh
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 10
3 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199
 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313
317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 4
39 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 56
9 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677
 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821
823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 9
53 967 971 977 983 991 997
jordthanasi@Jords-MacBook-Air ex15 % ./ex15.sh
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 10
3 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199
 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313
317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 4
39 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 56
9 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677
 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821
823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 9
53 967 971 977 983 991 997
jordthanasi@Jords-MacBook-Air ex15 % ./ex15.sh
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 10
3 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199
 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313
317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 4
39 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 56
9 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677
 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821
823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 9
53 967 971 977 983 991 997
jordthanasi@Jords-MacBook-Air ex15 %
```

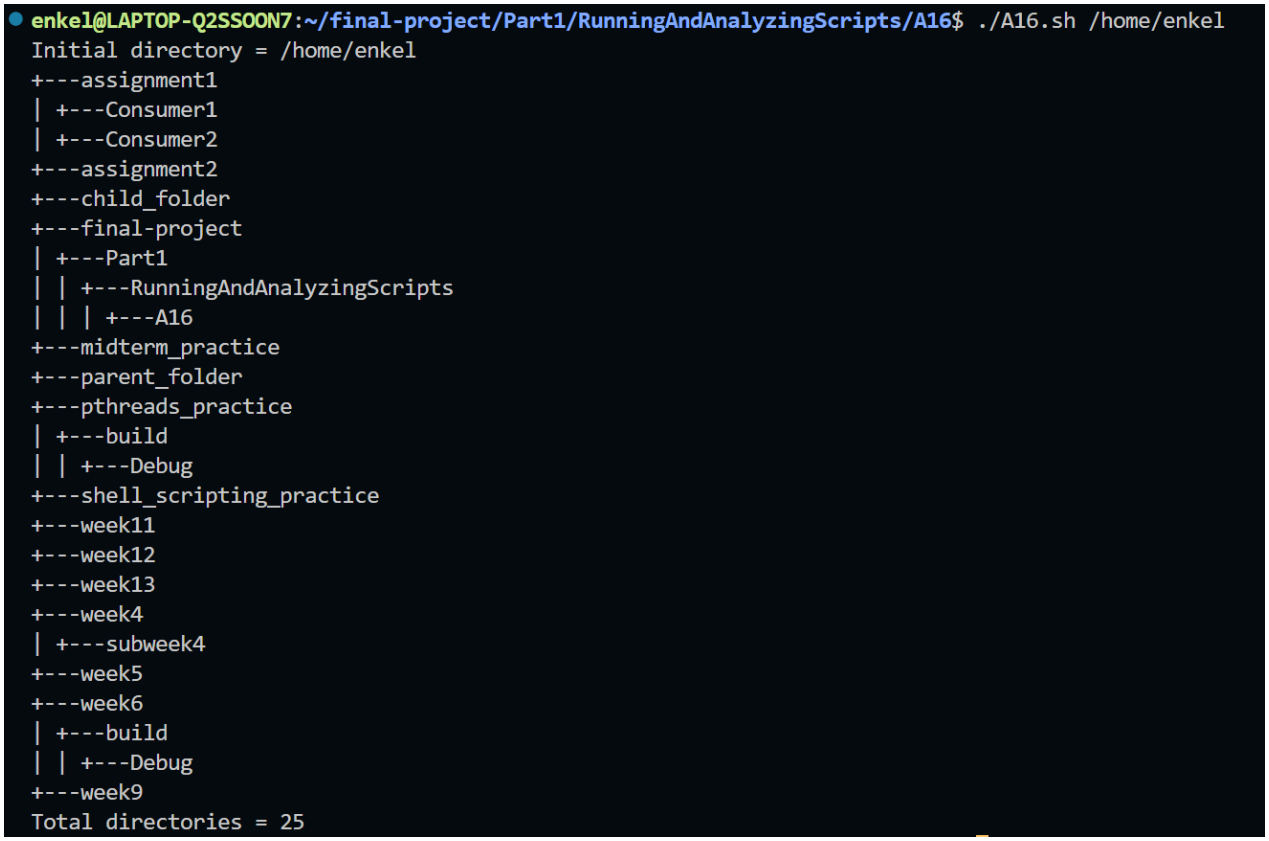
Rick Boivie's revision of Jordi Sanfeliu's *tree* script.

Example A-16. *tree*: Displaying a directory tree

This is a bash script that creates a visual directory tree, similar to the Unix tree command.
It recursively traverses directories starting from a given path (or current directory). It displays the directory structure. Counts and reports the total number of directories found. Handles symbolic links by showing their target information.

```
#!/bin/bash
# tree.sh
# Written by Rick Boivie.
# Used with permission.
# This is a revised and simplified version of a script
#+ by Jordi Sanfeliu (the original author), and patched by Ian Kjos.
# This script replaces the earlier version used in
#+ previous releases of the Advanced Bash Scripting Guide.
# Copyright (c) 2002, by Jordi Sanfeliu, Rick Boivie, and Ian Kjos.
# ==> Comments added by the author of this document.

# This function performs the directory traversal
search() {
    for dir in `echo *`; do
        # ==> `echo *` lists all the files in current working directory,
        #+ ==> without line breaks.
        # ==> Similar effect to for dir in *
        # ==> but "dir in `echo *`" will not handle filenames with blanks.

        if [ -d "$dir" ]; then # ==> If it is a directory (-d)...
            zz=0 # ==> Temp variable, keeping track of directory level.

            while [ $zz != $1 ]; do # Keep track of inner nested loop.
                echo -n "| " # ==> Display vertical connector symbol,
                             # ==> with 2 spaces & no line feed
                             # in order to indent.
                zz=`expr $zz + 1` # ==> Increment zz.
            done

            if [ -L "$dir" ]; then # ==> If directory is a symbolic link...
                echo "+---$dir" `ls -l $dir | sed 's/^.*'$dir' //'`
                # ==> Display horiz. connector and list directory name, but...
                # ==> delete date/time part of long listing.
            else
                echo "+---$dir" # ==> Display horizontal connector symbol...
                             # ==> and print directory name.
                numdirs=`expr $numdirs + 1` # ==> Increment directory count.

                if cd "$dir"; then # ==> If can move to subdirectory...
                    search `expr $1 + 1` # with recursion ;-)
                                         # ==> Function calls itself.
                    cd ..
```

```
                fi
            fi
        fi
    done
}

if [ $# != 0 ]; then
    cd $1 # Move to indicated directory.
    #else # stay in current directory
fi

echo "Initial directory = `pwd`"
numdirs=0
search 0
echo "Total directories = $numdirs"
exit 0
```



Patsie's version of a directory *tree* script.

Example A-17. *tree2*: Alternate directory tree script

This code is a disk usage analyzer that creates a visual tree showing the largest files

and directories with their sizes.

Core Functions:

- **dot()**: Formats numbers with commas (e.g., 1234 → 1,234) for readability
- **tree()**: Recursive function that builds the visual tree structure

Main Logic:

1. **Input validation**: Checks if a directory argument is provided
2. **Disk analysis**: Uses du -akx to get sizes of all files/directories
3. **Tree generation**: Recursively displays the TOP 5 largest items at each level
4. **Visual formatting**: Creates tree with +- connectors and proper indentation

```
#!/bin/bash
# tree2.sh
# Lightly modified/reformatted by ABS Guide author.
# Included in ABS Guide with permission of script author (thanks!).
## Recursive file/dirsize checking script, by Patsie
##
## This script builds a list of files/directories and their size (du -akx)
## and processes this list to a human readable tree shape
## The 'du -akx' is only as good as the permissions the owner has.
## So preferably run as root* to get the best results, or use only on
## directories for which you have read permissions. Anything you can't
## read is not in the list.
#* ABS Guide author advises caution when running scripts as root!

########## THIS IS CONFIGURABLE ##########
TOP=5                    # Top 5 biggest (sub)directories.
MAXRECURS=5             # Max 5 subdirectories/recursions deep.
E_BL=80                 # Blank line already returned.
E_DIR=81                # Directory not specified.

########## DON'T CHANGE ANYTHING BELOW THIS LINE ##########
PID=$$                  # Our own process ID.
SELF=`basename $0`      # Our own program name.
TMP="/tmp/${SELF}.${PID}.tmp"  # Temporary 'du' result.

# Convert number to dotted thousand.
function dot {
    echo " $*" | \
        sed -e :a -e 's/\(.*[0-9]\)\([0-9]\{3\}\)/\1,\2/;ta' | \
        tail -c 12
}

# Usage: tree <recursion> <indent prefix> <min size> <directory>
function tree {
    recurs="$1"         # How deep nested are we?
    prefix="$2"         # What do we display before file/dirname?
```

```
    minsize="$3"       # What is the minimum file/dirsize?
    dirname="$4"       # Which directory are we checking?

    # Get ($TOP) biggest subdirs/subfiles from TMP file.
    LIST=`egrep "[[:space:]]${dirname}/[^/]*$" "$TMP" | \
          awk '{if($1>'$minsize') print;}' | \
          sort -nr | \
          head -$TOP`

    [ -z "$LIST" ] && return  # Empty list, then go back.

    cnt=0
    num=`echo "$LIST" | wc -l`  # How many entries in the list.

    ## Main loop
    echo "$LIST" | while read size name; do
        ((cnt+=1))  # Count entry number.
        bname=`basename "$name"`  # We only need a basename of the entry.

        [ -d "$name" ] && bname="$bname/"
        # If it's a directory, append a slash.

        echo "`dot $size`$prefix +-$bname"
        # Display the result.

        # Call ourself recursively if it's a directory
        #+ and we're not nested too deep ($MAXRECURS).
        # The recursion goes up: $((recurs+1))
        # The prefix gets a space if it's the last entry,
        #+ or a pipe if there are more entries.
        # The minimum file/dirsize becomes
        #+ a tenth of his parent: $((size/10)).
        # Last argument is the full directory name to check.
        if [ -d "$name" -a $recurs -lt $MAXRECURS ]; then
            [ $cnt -lt $num ] \
                || (tree $((recurs+1)) "$prefix " $((size/10)) "$name") \
                && (tree $((recurs+1)) "$prefix |" $((size/10)) "$name")
        fi
    done

    [ $? -eq 0 ] && echo " $prefix"
    # Every time we jump back add a 'blank' line.
    return $E_BL
    # We return 80 to tell we added a blank line already.
}

### ###
### main program ###
### ###

rootdir="$@"
[ -d "$rootdir" ] || {
    echo "$SELF: Usage: $SELF <directory>" >&2
    exit $E_DIR
```

```
}
# We should be called with a directory name.

echo "Building inventory list, please wait ..."
# Show "please wait" message.

du -akx "$rootdir" 1>"$TMP" 2>/dev/null
# Build a temporary list of all files/dirs and their size.

size=`tail -1 "$TMP" | awk '{print $1}'`
# What is our rootdirectory's size?

echo "`dot $size` $rootdir"
# Display rootdirectory's entry.

tree 0 "" 0 "$rootdir"
# Display the tree below our rootdirectory.

rm "$TMP" 2>/dev/null
# Clean up TMP file.

exit $?
```

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/RunningAndAnalyzingScripts/A17$ ./A17.sh /home/enkel/final-project
Building inventory list, please wait ...
 108 /home/enkel/final-project
 104 +-Part1/
 100  +-RunningAndAnalyzingScripts/
 88   +-A16/
 80    +-Screenshot 2025-06-14 115558.png
```

Noah Friedman permitted use of his *string function* script. It essentially reproduces some of the *C*-library string manipulation functions.

Example A-18. *string functions*: C-style string functions

This is a bash string manipulation library that replicates C's string.h functions in pure bash.

What it does:

- **String operations:** Concatenation (strcat, strncat), comparison (strcmp, strncmp), length (strlen)
- **Pattern matching:** Find characters (strspn, strcspn), substrings (strstr), truncation (strtrunc)
- **Advanced techniques:** Uses indirect variable expansion, parameter substitution, and pattern matching

Its core technique uses indirect variable expansion (${!var}) to manipulate variables

by name, making it work like C functions that modify variables through pointers.

```
#!/bin/bash
# string.bash --- bash emulation of string(3) library routines
# Author: Noah Friedman <friedman@prep.ai.mit.edu>
# ==> Used with his kind permission in this document.
# Created: 1992-07-01
# Last modified: 1993-09-29
# Public domain
# Conversion to bash v2 syntax done by Chet Ramey
# Commentary:
# Code:

#:docstring strcat:
# Usage: strcat s1 s2
#
# Strcat appends the value of variable s2 to variable s1.
#
# Example:
# a="foo"
# b="bar"
# strcat a b
# echo $a
# => foobar
#
#:end docstring:

###;;;autoload ==> Autoloading of function commented out.
function strcat() {
    local s1_val s2_val
    s1_val=${!1}  # indirect variable expansion
    s2_val=${!2}
    eval "$1"=\'"${s1_val}${s2_val}"\'
    # ==> eval $1='${s1_val}${s2_val}' avoids problems,
    # ==> if one of the variables contains a single quote.
}

#:docstring strncat:
# Usage: strncat s1 s2 $n
#
# Line strcat, but strncat appends a maximum of n characters from the value
# of variable s2. It copies fewer if the value of variabl s2 is shorter
# than n characters. Echoes result on stdout.
#
# Example:
# a=foo
# b=barbaz
# strncat a b 3
# echo $a
# => foobar
#
#:end docstring:
```

```
###;;;autoload
function strncat() {
    local s1="$1"
    local s2="$2"
    local -i n="$3"
    local s1_val s2_val

    s1_val=${!s1}  # ==> indirect variable expansion
    s2_val=${!s2}

    if [ ${#s2_val} -gt ${n} ]; then
        s2_val=${s2_val:0:$n}  # ==> substring extraction
    fi

    eval "$s1"=\'"${s1_val}${s2_val}"\'
    # ==> eval $1='${s1_val}${s2_val}' avoids problems,
    # ==> if one of the variables contains a single quote.
}

#:docstring strcmp:
# Usage: strcmp $s1 $s2
#
# Strcmp compares its arguments and returns an integer less than, equal to,
# or greater than zero, depending on whether string s1 is lexicographically
# less than, equal to, or greater than string s2.
#:end docstring:

###;;;autoload
function strcmp() {
    [ "$1" = "$2" ] && return 0
    [ "${1}" '<' "${2}" ] > /dev/null && return -1
    return 1
}

#:docstring strncmp:
# Usage: strncmp $s1 $s2 $n
#
# Like strcmp, but makes the comparison by examining a maximum of n
# characters (n less than or equal to zero yields equality).
#:end docstring:

###;;;autoload
function strncmp() {
    if [ -z "${3}" -o "${3}" -le "0" ]; then
        return 0
    fi

    if [ ${3} -ge ${#1} -a ${3} -ge ${#2} ]; then
        strcmp "$1" "$2"
        return $?
    else
        s1=${1:0:$3}
        s2=${2:0:$3}
        strcmp $s1 $s2
```

```
        return $?
    fi
}

#:docstring strlen:
# Usage: strlen s
#
# Strlen returns the number of characters in string literal s.
#:end docstring:

###;;;autoload
function strlen() {
    eval echo "\${#${1}}"
    # ==> Returns the length of the value of the variable
    # ==> whose name is passed as an argument.
}

#:docstring strspn:
# Usage: strspn $s1 $s2
#
# Strspn returns the length of the maximum initial segment of string s1,
# which consists entirely of characters from string s2.
#:end docstring:

###;;;autoload
function strspn() {
    # Unsetting IFS allows whitespace to be handled as normal chars.
    local IFS=
    local result="${1%%[!${2}]*}"
    echo ${#result}
}

#:docstring strcspn:
# Usage: strcspn $s1 $s2
#
# Strcspn returns the length of the maximum initial segment of string s1,
# which consists entirely of characters not from string s2.
#:end docstring:

###;;;autoload
function strcspn() {
    # Unsetting IFS allows whitspace to be handled as normal chars.
    local IFS=
    local result="${1%%[${2}]*}"
    echo ${#result}
}

#:docstring strstr:
# Usage: strstr s1 s2
#
# Strstr echoes a substring starting at the first occurrence of string s2 in
# string s1, or nothing if s2 does not occur in the string. If s2 points to
# a string of zero length, strstr echoes s1.
#:end docstring:
```

```
###;;;autoload
function strstr() {
    # if s2 points to a string of zero length, strstr echoes s1
    [ ${#2} -eq 0 ] && { echo "$1"; return 0; }

    # strstr echoes nothing if s2 does not occur in s1
    case "$1" in
        *$2*) ;;
        *) return 1;;
    esac

    # use the pattern matching code to strip off the match and everything
    # following it
    first=${1/$2*/}
    # then strip off the first unmatched portion of the string
    echo "${1##$first}"
}

#:docstring strtok:
# Usage: strtok s1 s2
#
# Strtok considers the string s1 to consist of a sequence of zero or more
# text tokens separated by spans of one or more characters from the
# separator string s2. The first call (with a non-empty string s1
# specified) echoes a string consisting of the first token on stdout. The
# function keeps track of its position in the string s1 between separate
# calls, so that subsequent calls made with the first argument an empty
# string will work through the string immediately following that token. In
# this way subsequent calls will work through the string s1 until no tokens
# remain. The separator string s2 may be different from call to call.
# When no token remains in s1, an empty value is echoed on stdout.
#:end docstring:

###;;;autoload
function strtok() {
    :
}

#:docstring strtrunc:
# Usage: strtrunc $n $s1 {$s2} {$...}
#
# Used by many functions like strncmp to truncate arguments for comparison.
# Echoes the first n characters of each string s1 s2 ... on stdout.
#:end docstring:

###;;;autoload
function strtrunc() {
    n=$1; shift
    for z; do
        echo "${z:0:$n}"
    done
}
```

```
# provide string
# string.bash ends here
# ================================================================================
#
# ==> Everything below here added by the document author.
# ==> Suggested use of this script is to delete everything below here,
# ==> and "source" this file into your own scripts.

# strcat
string0=one
string1=two
echo
echo "Testing \"strcat\" function:"
echo "Original \"string0\" = $string0"
echo "\"string1\" = $string1"
strcat string0 string1
echo "New \"string0\" = $string0"
echo

# strlen
echo
echo "Testing \"strlen\" function:"
str=123456789
echo "\"str\" = $str"
echo -n "Length of \"str\" = "
strlen str
echo

# Exercise:
# --------
# Add code to test all the other string functions above.
exit 0
```

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/RunningAndAnalyzingScripts/A18$ ./A18.sh

Testing "strcat" function:
Original "string0" = one
"string1" = two
New "string0" = onetwo


Testing "strlen" function:
"str" = 123456789
Length of "str" = 9
```

Michael Zick's complex array example uses the [md5sum] check sum command to encode directory information.

Example A-19. Directory information

This is a comprehensive directory analysis and file system information script that implements

a content-based file identification system.

Functions:

- **ListDirectory():** Gets detailed file listings using ls with inode, permissions, timestamps, etc.
- **IndexList():** Parses the directory listing and creates an index mapping for efficient access
- **DigestFile():** Generates MD5 checksums to identify files by content rather than name
- **LocateFile():** Extracts file location metadata using stat command
- **ListArray():** Utility function to display array contents in formatted output

Main Execution Flow:

1. Analyze current directory
2. Generate directory checksum
3. List directory contents
4. Index the contents
5. Generate content checksum
6. Analyze file location

This script is designed for: removing duplicate files based on the content, directory structure analysis, file system forensics and file management.

```
#! /bin/bash
# directory-info.sh
# Parses and lists directory information.
# NOTE: Change lines 273 and 353 per "README" file.
# Michael Zick is the author of this script.
# Used here with his permission.

# Controls
# If overridden by command arguments, they must be in the order:
# Arg1: "Descriptor Directory"
# Arg2: "Exclude Paths"
# Arg3: "Exclude Directories"
#
# Environment Settings override Defaults.
# Command arguments override Environment Settings.

# Default location for content addressed file descriptors.
MD5UCFS=${1:-${MD5UCFS:-'/tmpfs/ucfs'}}

# Directory paths never to list or enter
declare -a \
EXCLUDE_PATHS=${2:-${EXCLUDE_PATHS:-'(/proc /dev /devfs /tmpfs)'}}

# Directories never to list or enter
```

```
declare -a \
EXCLUDE_DIRS=${3:-${EXCLUDE_DIRS:-'(ucfs lost+found tmp wtmp)'}}

# Files never to list or enter
declare -a \
EXCLUDE_FILES=${3:-${EXCLUDE_FILES:-'(core "Name with Spaces")'}}

# Here document used as a comment block.
: <<LSfieldsDoc
# # # # # List Filesystem Directory Information # # # # #
#
# ListDirectory "FileGlob" "Field-Array-Name"
# or
# ListDirectory -of "FileGlob" "Field-Array-Filename"
# '-of' meaning 'output to filename'
# # # # #
String format description based on: ls (GNU fileutils) version 4.0.36
Produces a line (or more) formatted:
inode permissions hard-links owner group ...
32736 -rw------- 1 mszick mszick
size day month date hh:mm:ss year path
2756608 Sun Apr 20 08:53:06 2003 /home/mszick/core
Unless it is formatted:
inode permissions hard-links owner group ...
266705 crw-rw---- 1 root uucp

major minor day month date hh:mm:ss year path
4, 68 Sun Apr 20 09:27:33 2003 /dev/ttyS4
NOTE: that pesky comma after the major number
NOTE: the 'path' may be multiple fields:
/home/mszick/core
/proc/982/fd/0 -> /dev/null
/proc/982/fd/1 -> /home/mszick/.xsession-errors
/proc/982/fd/13 -> /tmp/tmpfZVVOCs (deleted)
/proc/982/fd/7 -> /tmp/kde-mszick/ksycoca
/proc/982/fd/8 -> socket:[11586]
/proc/982/fd/9 -> pipe:[11588]
If that isn't enough to keep your parser guessing,
either or both of the path components may be relative:
../Built-Shared -> Built-Static
../linux-2.4.20.tar.bz2 -> ../../../SRCS/linux-2.4.20.tar.bz2
The first character of the 11 (10?) character permissions field:
's' Socket
'd' Directory
'b' Block device
'c' Character device
'l' Symbolic link
NOTE: Hard links not marked - test for identical inode numbers
on identical filesystems.
All information about hard linked files are shared, except
for the names and the name's location in the directory system.
NOTE: A "Hard link" is known as a "File Alias" on some systems.
'-' An undistingushed file
Followed by three groups of letters for: User, Group, Others
```

```
Character 1: '-' Not readable; 'r' Readable
Character 2: '-' Not writable; 'w' Writable
Character 3, User and Group: Combined execute and special
'-' Not Executable, Not Special
'x' Executable, Not Special
's' Executable, Special
'S' Not Executable, Special
Character 3, Others: Combined execute and sticky (tacky?)
'-' Not Executable, Not Tacky
'x' Executable, Not Tacky
't' Executable, Tacky
'T' Not Executable, Tacky
Followed by an access indicator
Haven't tested this one, it may be the eleventh character
or it may generate another field
' ' No alternate access
'+' Alternate access
LSfieldsDoc

# This function lists the directory contents
ListDirectory() {
    local -a T
    local -i of=0 # Default return in variable
    # OLD_IFS=$IFS # Using BASH default ' \t\n'

    case "$#" in
        3) case "$1" in
            -of) of=1 ; shift ;;
            * ) return 1 ;;
        esac ;;
        2) : ;; # Poor man's "continue"
        *) return 1 ;;
    esac

    # NOTE: the (ls) command is NOT quoted (")
    T=( $(ls --inode --ignore-backups --almost-all --directory \
          --full-time --color=none --time=status --sort=none \
          --format=long $1) )

    case $of in
        # Assign T back to the array whose name was passed as $2
        0) eval $2=\( \"\$\{T\[@\]\}\" \) ;;
        # Write T into filename passed as $2
        1) echo "${T[@]}" > "$2" ;;
    esac
    return 0
}

# # # # # Is that string a legal number? # # # # #
#
# IsNumber "Var"
# # # # # There has to be a better way, sigh...
#Fucntion to check if a string is a number
IsNumber() {
```

```
    local -i int
    if [ $# -eq 0 ]; then
        return 1
    else
        (let int=$1) 2>/dev/null
        return $? # Exit status of the let thread
    fi
}

# # # # # Index Filesystem Directory Information # # # # #
#
# IndexList "Field-Array-Name" "Index-Array-Name"
# or
# IndexList -if Field-Array-Filename Index-Array-Name
# IndexList -of Field-Array-Name Index-Array-Filename
# IndexList -if -of Field-Array-Filename Index-Array-Filename
# # # # #
: <<IndexListDoc
Walk an array of directory fields produced by ListDirectory

Having suppressed the line breaks in an otherwise line oriented
report, build an index to the array element which starts each line.
Each line gets two index entries, the first element of each line
(inode) and the element that holds the pathname of the file.
The first index entry pair (Line-Number==0) are informational:
Index-Array-Name[0] : Number of "Lines" indexed
Index-Array-Name[1] : "Current Line" pointer into Index-Array-Name
The following index pairs (if any) hold element indexes into
the Field-Array-Name per:
Index-Array-Name[Line-Number * 2] : The "inode" field element.
NOTE: This distance may be either +11 or +12 elements.
Index-Array-Name[(Line-Number * 2) + 1] : The "pathname" element.
NOTE: This distance may be a variable number of elements.
Next line index pair for Line-Number+1.
IndexListDoc

# Function to index the directory information
IndexList() {
    local -a LIST # Local of listname passed
    local -a -i INDEX=( 0 0 ) # Local of index to return
    local -i Lidx Lcnt
    local -i if=0 of=0 # Default to variable names

    case "$#" in # Simplistic option testing
        0) return 1 ;;
        1) return 1 ;;
        2) : ;; # Poor man's continue
        3) case "$1" in
            -if) if=1 ;;
            -of) of=1 ;;
            * ) return 1 ;;
        esac ; shift ;;
        4) if=1 ; of=1 ; shift ; shift ;;
        *) return 1
```

```
    esac

    # Make local copy of list
    case "$if" in
        0) eval LIST=\( \"\$\{$1\[@\]\}\" \) ;;
        1) LIST=( $(cat $1) ) ;;
    esac

    # Grok (grope?) the array
    Lcnt=${#LIST[@]}
    Lidx=0
    until (( Lidx >= Lcnt )); do
        if IsNumber ${LIST[$Lidx]}; then
            local -i inode name
            local ft

            inode=Lidx
            local m=${LIST[$Lidx+2]} # Hard Links field
            ft=${LIST[$Lidx+1]:0:1} # Fast-Stat
            case $ft in
                b) ((Lidx+=12)) ;; # Block device
                c) ((Lidx+=12)) ;; # Character device
                *) ((Lidx+=11)) ;; # Anything else
            esac
            name=Lidx
            case $ft in
                -) ((Lidx+=1)) ;; # The easy one
                b) ((Lidx+=1)) ;; # Block device
                c) ((Lidx+=1)) ;; # Character device
                d) ((Lidx+=1)) ;; # The other easy one
                l) ((Lidx+=3)) ;; # At LEAST two more fields

                # A little more elegance here would handle pipes,
                #+ sockets, deleted files - later.

                *) until IsNumber ${LIST[$Lidx]} || ((Lidx >= Lcnt)); do
                    ((Lidx+=1))
                done
                ;; # Not required
            esac
            INDEX[${#INDEX[*]}]=$inode
            INDEX[${#INDEX[*]}]=$name
            INDEX[0]=${INDEX[0]}+1 # One more "line" found

            # echo "Line: ${INDEX[0]} Type: $ft Links: $m Inode: \
            # ${LIST[$inode]} Name: ${LIST[$name]}"
        else
            ((Lidx+=1))
        fi
    done

    case "$of" in
        0) eval $2=\( \"\$\{INDEX\[@\]\}\" \) ;;
        1) echo "${INDEX[@]}" > "$2" ;;
```

```
    esac
    return 0 # What could go wrong?
}

# # # # # Content Identify File # # # # #
#
# DigestFile Input-Array-Name Digest-Array-Name
# or
# DigestFile -if Input-FileName Digest-Array-Name
# # # # #
# Here document used as a comment block.
: <<DigestFilesDoc
The key (no pun intended) to a Unified Content File System (UCFS)
is to distinguish the files in the system based on their content.
Distinguishing files by their name is just so 20th Century.
The content is distinguished by computing a checksum of that content.
This version uses the md5sum program to generate a 128 bit checksum

representative of the file's contents.
There is a chance that two files having different content might
generate the same checksum using md5sum (or any checksum). Should
that become a problem, then the use of md5sum can be replace by a
cyrptographic signature. But until then...
The md5sum program is documented as outputting three fields (and it
does), but when read it appears as two fields (array elements). This
is caused by the lack of whitespace between the second and third field.
So this function gropes the md5sum output and returns:
[0] 32 character checksum in hexidecimal (UCFS filename)
[1] Single character: ' ' text file, '*' binary file
[2] Filesystem (20th Century Style) name
Note: That name may be the character '-' indicating STDIN read.
DigestFilesDoc

# Function that generates a cheksum for a file
DigestFile() {
    local if=0 # Default, variable name
    local -a T1 T2

    case "$#" in
        3) case "$1" in
            -if) if=1 ; shift ;;
            * ) return 1 ;;
        esac ;;
        2) : ;; # Poor man's "continue"
        *) return 1 ;;
    esac

    case $if in
        0) eval T1=\( \"\$\{$1\[@\]\}\" \)
           T2=( $(echo ${T1[@]} | md5sum -) )
        ;;
        1) T2=( $(md5sum $1) )
        ;;
    esac
```

```
    case ${#T2[@]} in
        0) return 1 ;;
        1) return 1 ;;
        2) case ${T2[1]:0:1} in # SanScrit-2.0.5
            \*) T2[${#T2[@]}]=${T2[1]:1}
                T2[1]=\*
            ;;
            *) T2[${#T2[@]}]=${T2[1]}
               T2[1]=" "
            ;;
        esac
        ;;
        3) : ;; # Assume it worked
        *) return 1 ;;
    esac

    local -i len=${#T2[0]}
    if [ $len -ne 32 ]; then return 1; fi
    eval $2=\( \"\$\{T2\[@\]\}\" \)
}

# # # # # Locate File # # # # #
#
# LocateFile [-l] FileName Location-Array-Name
# or
# LocateFile [-l] -of FileName Location-Array-FileName
# # # # #
# A file location is Filesystem-id and inode-number
# Here document used as a comment block.
: <<StatFieldsDoc
Based on stat, version 2.2
stat -t and stat -lt fields
[0] name
[1] Total size
File - number of bytes
Symbolic link - string length of pathname
[2] Number of (512 byte) blocks allocated
[3] File type and Access rights (hex)
[4] User ID of owner
[5] Group ID of owner
[6] Device number
[7] Inode number
[8] Number of hard links
[9] Device type (if inode device) Major
[10] Device type (if inode device) Minor
[11] Time of last access
May be disabled in 'mount' with noatime
atime of files changed by exec, read, pipe, utime, mknod

(mmap?)

atime of directories changed by addition/deletion of files
[12] Time of last modification
```

```
mtime of files changed by write, truncate, utime, mknod
mtime of directories changed by addtition/deletion of files
[13] Time of last change
ctime reflects time of changed inode information (owner,

group

permissions, link count

-*-*- Per:
Return code: 0
Size of array: 14
Contents of array
Element 0: /home/mszick
Element 1: 4096
Element 2: 8
Element 3: 41e8
Element 4: 500
Element 5: 500
Element 6: 303
Element 7: 32385

Element 8: 22
Element 9: 0
Element 10: 0
Element 11: 1051221030
Element 12: 1051214068
Element 13: 1051214068
For a link in the form of linkname -> realname
stat -t linkname returns the linkname (link) information
stat -lt linkname returns the realname information
stat -tf and stat -ltf fields
[0] name
[1] ID-0? # Maybe someday, but Linux stat structure
[2] ID-0? # does not have either LABEL nor UUID
# fields, currently information must come
# from file-system specific utilities

These will be munged into:
[1] UUID if possible
[2] Volume Label if possible
Note: 'mount -l' does return the label and could return the UUID
[3] Maximum length of filenames
[4] Filesystem type
[5] Total blocks in the filesystem
[6] Free blocks
[7] Free blocks for non-root user(s)
[8] Block size of the filesystem
[9] Total inodes
[10] Free inodes
-*-*- Per:
Return code: 0
Size of array: 11
Contents of array
```

```
Element 0: /home/mszick
Element 1: 0
Element 2: 0
Element 3: 255
Element 4: ef53
Element 5: 2581445
Element 6: 2277180
Element 7: 2146050
Element 8: 4096
Element 9: 1311552
Element 10: 1276425
StatFieldsDoc

# LocateFile [-l] FileName Location-Array-Name
# LocateFile [-l] -of FileName Location-Array-FileName
# Function that locaets a file
LocateFile() {
    local -a LOC LOC1 LOC2
    local lk="" of=0

    case "$#" in
        0) return 1 ;;
        1) return 1 ;;
        2) : ;;
        *) while (( "$#" > 2 )); do
            case "$1" in
                -l) lk=-1 ;;
                -of) of=1 ;;
                *) return 1 ;;
            esac
            shift
        done ;;
    esac

    # More Sanscrit-2.0.5
    # LOC1=( $(stat -t $lk $1) )
    # LOC2=( $(stat -tf $lk $1) )
    # Uncomment above two lines if system has "stat" command installed.
    LOC=( ${LOC1[@]:0:1} ${LOC1[@]:3:11}
          ${LOC2[@]:1:2} ${LOC2[@]:4:1} )

    case "$of" in
        0) eval $2=\( \"\$\{LOC\[@\]\}\" \) ;;
        1) echo "${LOC[@]}" > "$2" ;;
    esac
    return 0
    # Which yields (if you are lucky, and have "stat" installed)
    # -*-*- Location Discriptor -*-*-
    # Return code: 0
    # Size of array: 15
    # Contents of array
    # Element 0: /home/mszick 20th Century name
    # Element 1: 41e8 Type and Permissions
    # Element 2: 500 User
```

```
    # Element 3: 500 Group
    # Element 4: 303 Device
    # Element 5: 32385 inode
    # Element 6: 22 Link count
    # Element 7: 0 Device Major
    # Element 8: 0 Device Minor
    # Element 9: 1051224608 Last Access
    # Element 10: 1051214068 Last Modify
    # Element 11: 1051214068 Last Status
    # Element 12: 0 UUID (to be)
    # Element 13: 0 Volume Label (to be)
    # Element 14: ef53 Filesystem type
}

# And then there was some test code
# This function lists the contents of an array
ListArray() { # ListArray Name
    local -a Ta
    eval Ta=\( \"\$\{$1\[@\]\}\" \)
    echo
    echo "-*-*- List of Array -*-*-"
    echo "Size of array $1: ${#Ta[*]}"
    echo "Contents of array $1:"
    for (( i=0 ; i<${#Ta[*]} ; i++ )); do
        echo -e "\tElement $i: ${Ta[$i]}"
    done
    return 0
}

# Declare and list the directory
declare -a CUR_DIR
# For small arrays
ListDirectory "${PWD}" CUR_DIR
ListArray CUR_DIR

# Generate the checksum of the directory
declare -a DIR_DIG
DigestFile CUR_DIR DIR_DIG
echo "The new \"name\" (checksum) for ${CUR_DIR[9]} is ${DIR_DIG[0]}"

# List the contents of the directory
declare -a DIR_ENT
# BIG_DIR # For really big arrays - use a temporary file in ramdisk
# BIG-DIR # ListDirectory -of "${CUR_DIR[11]}/*" "/tmpfs/junk2"
ListDirectory "${CUR_DIR[11]}/*" DIR_ENT

# Index the directory contents
declare -a DIR_IDX
# BIG-DIR # IndexList -if "/tmpfs/junk2" DIR_IDX
IndexList DIR_ENT DIR_IDX

# Generate the checksum of the directory contents
declare -a IDX_DIG
# BIG-DIR # DIR_ENT=( $(cat /tmpfs/junk2) )
```

```
# BIG-DIR # DigestFile -if /tmpfs/junk2 IDX_DIG
DigestFile DIR_ENT IDX_DIG
# Small (should) be able to parallize IndexList & DigestFile
# Large (should) be able to parallize IndexList & DigestFile & the assignment
echo "The \"name\" (checksum) for the contents of ${PWD} is ${IDX_DIG[0]}"

# Locate the file and list the details of the file
declare -a FILE_LOC
LocateFile ${PWD} FILE_LOC
ListArray FILE_LOC

exit 0
```

Stéphane Chazelas demonstrates object-oriented programming in a Bash script. Mariusz Gniazdowski contributed a hash library for use in scripts.

Example A-20. Library of hash functions

This is a bash hash table (associative array) library that emulates hash/dictionary functionality in bash using global variables. It creates hash tables by generating global variables with naming pattern: __hash__hashname_keyname. It also provides hash-like operations similar to other programming languages.

Main Functions:

- **hash_set:** Sets a key-value pair (hash[key] = value)
- **hash_get_into:** Gets a value and stores it in a variable

- **hash_echo:** Prints a hash value directly
- **hash_copy:** Copies value from one hash key to another
- **hash_dup:** Duplicates a value to multiple keys
- **hash_unset:** Deletes a hash key
- **hash_get_ref_into:** Gets the variable name that holds the value
- **hash_echo_ref:** Prints the variable name
- **hash_call:** Executes a function stored in a hash value
- **hash_is_set:** Checks if a key exists
- **hash_foreach:** Iterates over all keys and calls a function for each

Hash function code:

```
#!/bin/bash
# Hash:

# Hash function library
# Author: Mariusz Gniazdowski <mariusz.gn-at-gmail.com>
# Date: 2005-04-07
# Functions making emulating hashes in Bash a little less painful.
# Limitations:
# * Only global variables are supported.
# * Each hash instance generates one global variable per value.
# * Variable names collisions are possible
#+ if you define variable like __hash__hashname_key
# * Keys must use chars that can be part of a Bash variable name
#+ (no dashes, periods, etc.).
# * The hash is created as a variable:
# ... hashname_keyname
# So if somone will create hashes like:
# myhash_ + mykey = myhash__mykey
# myhash + _mykey = myhash__mykey
# Then there will be a collision.
# (This should not pose a major problem.)

Hash_config_varname_prefix=__hash__

# Emulates: hash[key]=value
#
# Params:
# 1 - hash
# 2 - key
# 3 - value
# Sets a global variable with name: __hash__hashname_keyname
```

```
function hash_set {
    eval "${Hash_config_varname_prefix}${1}_${2}=\"${3}\""
}

# Emulates: value=hash[key]
#
# Params:
# 1 - hash
# 2 - key
# 3 - value (name of global variable to set)
# The function gets the value of the variable and assigns it
# to the provided variable name.
function hash_get_into {
    eval "$3=\"\$${Hash_config_varname_prefix}${1}_${2}\""
}

# Emulates: echo hash[key]
#
# Params:
# 1 - hash
# 2 - key
# 3 - echo params (like -n, for example)
# This functuon simply prints the value
function hash_echo {
    eval "echo $3 \"\$${Hash_config_varname_prefix}${1}_${2}\""
}

# Emulates: hash1[key1]=hash2[key2]
#
# Params:
# 1 - hash1
# 2 - key1
# 3 - hash2
# 4 - key2
# This function copies the value from one hash to another.
function hash_copy {
    eval "${Hash_config_varname_prefix}${1}_${2}\
=\"\$${Hash_config_varname_prefix}${3}_${4}\""
}

# Emulates: hash[keyN-1]=hash[key2]=...hash[key1]
#
# Copies first key to rest of keys.
#
# Params:
# 1 - hash1
# 2 - key1
# 3 - key2
# . . .
# N - keyN
# This function copies the value from first key to all other keys.
function hash_dup {
    local hashName="$1" keyName="$2"
    shift 2
    until [ ${#} -le 0 ]; do
        eval "${Hash_config_varname_prefix}${hashName}_${1}\
=\"\$${Hash_config_varname_prefix}${hashName}_${keyName}\""
```

```
        shift;
    done;
}

# Emulates: unset hash[key]
#
# Params:
# 1 - hash
# 2 - key
# This function deletes the variable that holds the hash key.
function hash_unset {
    eval "unset ${Hash_config_varname_prefix}${1}_${2}"
}

# Emulates something similar to: ref=&hash[key]
#
# The reference is name of the variable in which value is held.
#
# Params:
# 1 - hash
# 2 - key
# 3 - ref - Name of global variable to set.
# THis function sets the name of the variable that holds the hash key
function hash_get_ref_into {
    eval "$3=\"${Hash_config_varname_prefix}${1}_${2}\""
}

# Emulates something similar to: echo &hash[key]
#
# That reference is name of variable in which value is held.
#
# Params:
# 1 - hash
# 2 - key
# 3 - echo params (like -n for example)
# This function prints the name of the variable that holds the hash key.
function hash_echo_ref {
    eval "echo $3 \"${Hash_config_varname_prefix}${1}_${2}\""
}

# Emulates something similar to: $$hash[key](param1, param2, ...)
#
# Params:
# 1 - hash
# 2 - key
# 3,4, ... - Function parameters
# It calls the function that is stored in the hash key and
# passes the parameters to it.
function hash_call {
    local hash key
    hash=$1
    key=$2
    shift 2
    eval "eval \"\$${Hash_config_varname_prefix}${hash}_${key}
\\\"\\\$@\\\"\""
}
```

```
# Emulates something similar to: isset(hash[key]) or hash[key]==NULL
#
# Params:
# 1 - hash
# 2 - key
# Returns:
# 0 - there is such key
# 1 - there is no such key
# Checks if the hash key is set.
function hash_is_set {
    eval "if [[ \"\${${Hash_config_varname_prefix}${1}_${2}-a}\" = \"a\" &&
\"\${${Hash_config_varname_prefix}${1}_${2}-b}\" = \"b\" ]]
then return 1; else return 0; fi"
}

# Emulates something similar to:
# foreach($hash as $key => $value) { fun($key,$value); }
#
# It is possible to write different variations of this function.
# Here we use a function call to make it as "generic" as possible.
#
# Params:
# 1 - hash
# 2 - function name
# This function iterates over all keys in the hash and calls
# the provided function with two parameters: key and value.
function hash_foreach {
    local keyname oldIFS="$IFS"
    IFS=' '
    for i in $(eval "echo \${!${Hash_config_varname_prefix}${1}_*}"); do
        keyname=$(eval "echo \${i##${Hash_config_varname_prefix}${1}_}")
        eval "$2 $keyname \"\$$i\""
    done
    IFS="$oldIFS"
}

# NOTE: In lines 103 and 116, ampersand changed.
# But, it doesn't matter, because these are comment lines anyhow
```

Test hash function script:

```
#!/bin/bash
# Source the hash library
source /home/enkel/final-project/Part1/RunningAndAnalyzingScripts/A20/A20.sh

# Your test commands
hash_set myhash mykey "myvalue"

hash_get_into myhash mykey result
echo $result

hash_echo myhash mykey

hash_set myhash mykey1 "myvalue1"
hash_set otherhash otherkey1 "othervalue1"

hash_copy myhash mykey1 otherhash otherkey1
```

```
hash_set myhash mykey2 "myvalue2"
hash_set myhash mykey3 "myvalue3"

hash_dup myhash mykey1 mykey2 mykey3

hash_unset myhash mykey

hash_get_ref_into myhash mykey1 ref
echo $ref

hash_echo_ref myhash mykey1

hash_set myhash myfunc "echo Testing hash"
hash_call myhash myfunc

hash_is_set myhash mykey1 && echo "Key is set" || echo "Key is not set"

myfunction() { echo "$1 = $2"; }
hash_foreach myhash myfunction
```

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/RunningAndAnalyzingScripts/A20$ ./test_hash.sh
myvalue
myvalue
__hash__myhash_mykey1
__hash__myhash_mykey1
Testing hash
Key is set
myfunc = echo Testing hash
mykey1 = othervalue1
mykey2 = othervalue1
mykey3 = othervalue1
```

2) Analyzing Scripts

Provide, in the same way as above a complete analysis examining the following scripts.

a) Run the following script, then explain what it does. Annotate the script and rewrite it in a more compact and elegant manner.

```
#!/bin/bash

MAX=10000


for((nr=1; nr<$MAX; nr++))
do

let "t1 = nr % 5"
if [ "$t1" -ne 3 ]
then
continue
fi

let "t2 = nr % 7"
if [ "$t2" -ne 4 ]
then
continue
fi

let "t3 = nr % 9"
if [ "$t3" -ne 5 ]
then
continue
fi

break # What happens when you comment out this line? Why?


done

echo "Number = $nr"


exit 0
```

This script searches for the first number meeting specific mathematical conditions:

- nr % 5 = 3 (remainder 3 when divided by 5)
- nr % 7 = 4 (remainder 4 when divided by 7)
- nr % 9 = 5 (remainder 5 when divided by 9)

If we remove the break line, the code will not stop when it finds the first number that meets the conditions, but will continue to iterate until the end, and will print the last number that meets the conditions(10’000).

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/a$ ./a.sh
Number = 158
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/a$ ./a.sh
Number = 10000
```

The improved version of the code:

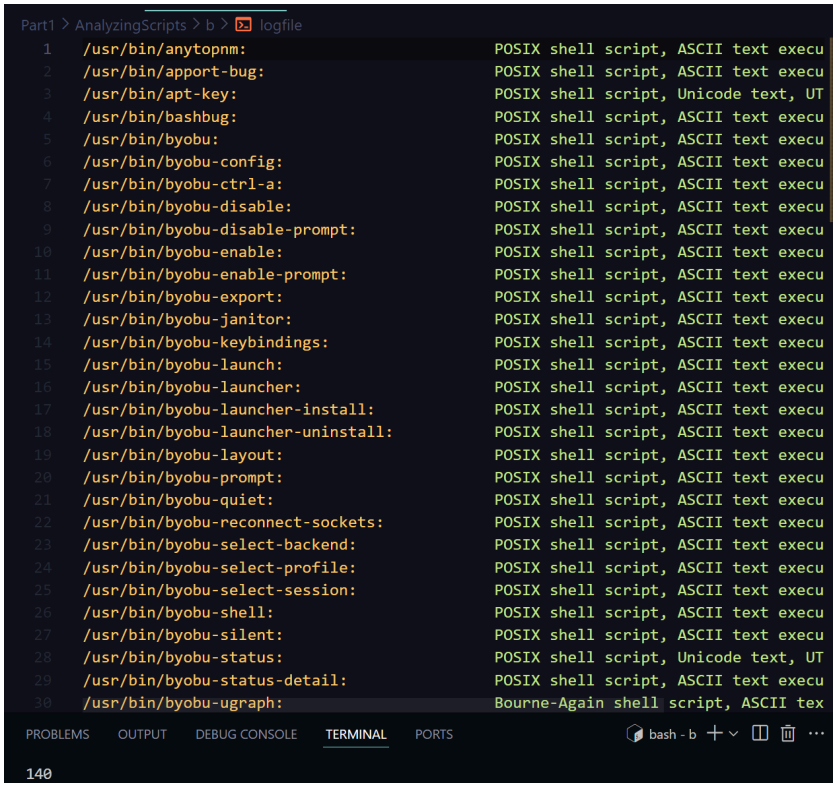
```
#!/bin/bash
# Find the first number from 1 to 10'000 that satisfies the conditions:
# - When divided by 5, the remainder is 3
# - When divided by 7, the remainder is 4
# - When divided by 9, the remainder is 5

for nr in {1..10000}; do  # Loop thought numbers from 1 to 10,000
    # Check if the number satisfies the conditions
    if (( nr%5==3 && nr%7==4 && nr%9==5 )); then
        echo "Number = $nr" #if the conditions are met, print the number
        exit 0              # and exit the script successfully
    fi
done
```

---

b) Explain what the following script does. It is really just a parameterized command-line pipe.

```
#!/bin/bash

DIRNAME=/usr/bin

FILETYPE="shell script"

LOGFILE=logfile

# Uses the file command to find the type of every file in the bin

# fgrep filters the results to show only shell scripts

# Then it uses tee to write the result to logfile

# It uses wc -l to count the number of lines in the output and

# prints it to the terminal

file "$DIRNAME"/* | fgrep "$FILETYPE" | tee $LOGFILE | wc -l

exit 0
```

This code is a shell script analyzer that scans bin directory to find and count shell scripts. It goes through every file in the bin directory and filters out the files that aren't shell scripts. It then prints all of the shell scripts to a new file named logfile and prints to the terminal the number of files.



---

c) Examine and explain the following script. For hints, you might refer to the listings for find and stat.

```
#!/bin/bash
# Author: Nathan Coulter
# This code is released to the public domain.
# The author gave permission to use this code snippet in the ABS Guide.
# Finds all the files in the current directory and prints their names
find -maxdepth 1 -type f -printf '%f\000' | {
  # Reads the names of the files, one by one
  while read -d $'\000'; do
    # Renames each file and adds the date and time of last modification
    mv "$REPLY" "$(date -d "$(stat -c '%y' "$REPLY")" '+%Y%m%d%H%M%S')-$REPLY"
  done
}

exit 0
# Warning: Test-drive this script in a "scratch" directory.
# It will somehow affect all the files there.
```

This code renames every file in the directory and adds timestamps for the last time the files in that

directory were modified.

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/c$ touch file1.txt
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/c$ touch file2.txt
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/c$ touch file3.txt
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/c$ touch file4.txt
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/c$ chmod 777 c.sh
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/c$ ls
c.sh  file1.txt  file2.txt  file3.txt  file4.txt
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/c$ ./c.sh
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/c$ ls
20250614162058-c.sh       20250614162311-file2.txt  20250614162321-file4.txt
20250614162304-file1.txt  20250614162317-file3.txt
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/c$
```

---

d) A reader sent in the following code snippet.

```
while read LINE
do
 echo $LINE
done < `tail -f /var/log/messages`
```

He wished to write a script tracking changes to the system log file, /var/log/messages. Unfortunately, the above code block hangs and does nothing useful. Why? Fix this so it does work. (Hint: rather than redirecting the stdin of the loop, try a pipe.)

The response to the terminal from the original code:

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/d$ ./d.sh
tail: cannot open '/var/log/messages' for reading: No such file or directory
tail: no files remaining
```

The fixed code:

```
#!/bin/bash

# The script reads the file provided and pipe output to while
# tail -f follows the file and shows new line as tehy are added
# The pipe sends each new line to the while loop
tail -f home/enkel/final-project/Part1/AnalyzingScripts/d` | while read LINE; do
  echo $LINE #Echo the current tail
done #End LOOP

exit 0
```

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/d$ chmod 777 d.sh
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/d$ ./d.sh
tail: cannot open '/var/log/messages' for reading: No such file or directory
tail: no files remaining
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/d
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/d
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/d$ ./d.sh
tail: cannot open 'home/enkel' for reading: No such file or directory
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/d$ ./d.sh
./d.sh: line 6: unexpected EOF while looking for matching ``'
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/d$ cd ..
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts$ ./d.sh
bash: ./d.sh: No such file or directory
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts$ cd d
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/d$ ./d.sh
./d.sh: line 6: unexpected EOF while looking for matching ``'
```

---

e) Analyze the following "one-liner" (here split into two lines for clarity) contributed by Rory Winston:

```
# The original script
# export SUM=0; for f in $(find src -name "*.java");
# do export SUM=$(($SUM + $(wc -l $f | awk '{ print $1 }'))); done; echo $SUM

# The modified script
# Initialize counter
export SUM=0

# Find all Java files
find src -name "*.java"

# Loop through files and count lines
for f in $(find src -name "*.java"); do
    export SUM=$(($SUM + $(wc -l $f | awk '{ print $1 }')))
done

# Display total
echo $SUM
```

Hint: First, break the script up into bite-sized sections. Then, carefully examine its use of double parentheses arithmetic, the export command, the find command, the wc command, and awk.

This code counts all the lines of every .java file in the src directory and its subdirectories.

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/e$ ls
e.sh  src
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/AnalyzingScripts/e$ ./e.sh
src/Example.java
src/main.java
35
```

---

3) Designing Scripts

1. Suppose in a directory that you wanted to change all the filenames ending in .f77 so that they instead ended in .f90. How would you go about it?
   If you're new to Unix you may need to find out how to change filenames. Typing apropos filename will list the programs and routines whose summaries mention filename. Alas, the most useful command isn't mentioned there. Typing apropos rename lists mv which is what you need. Maybe next you might try "mv *.f77 *.f90". Alas, this won't work - the shell will replace *.f77 by a list of filenames and the resulting command will fail. You need to use a loop of some sort. You also need a way to remove a suffix (look up basename) and how to add a new suffix. It's worth experimenting with a single name first. Do filename=test.f77 and see if you can produce a test.f90 string from $filename . One solution is

```
for filename in *.f77

do

 b=$(basename $filename .f77)

 mv $filename $b.f90
done
```

The script:

```
#!/bin/bash

# Rename all files in the directory that end with .f77, so that they end with .f90
for file in *.f77; do
    b=$(basename "$file" .f77)
    mv "$file" "$b.90"
done
```

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex1$ ls
ex1.sh  file1.f77  file2.f77  file3.f77
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex1$ ./ex1.sh
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex1$ ls
ex1.sh  file1.90  file2.90  file3.90
```

2. Change the args script supplied earlier so that if no argument is provided, "They are'' isn't printed, and if exactly 1 argument is provided, "... 1 argument'' rather than "... 1 arguments'' is printed (use if)

```
#!/bin/bash

# Change the args script supplied earlier so that if no argument is provided,
# "They are'' isn't printed,  and if exactly 1 argument is provided,
# "... 1 argument'' rather than "... 1 arguments'' is printed  (use if)

if [ $# -eq 0 ]; then
    echo "There are no arguments"
elif [ $# -eq 1 ]; then
    echo "There is 1 argument"
else
    echo "There are $# arguments"
fi
```

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex2$ chmod 777 ex2.sh
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex2$ ./ex2.sh
There are no arguments
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex2$ ./ex2.sh arg
There is 1 argument
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex2$ ./ex2.sh arg1 arg2 arg3
There are 3 arguments
```

3. Write a shell script that given a person's uid, tells you how many times that person is logged on. (who, grep, wc)

```
#!/bin/bash

#  Write a shell script that given a person's uid,
# tells you how many times that person is logged on.  (who, grep, wc)

if [ $# -eq 0 ]; then
    echo "Usage: $0 <uid>"
    exit 1
fi

uid=$1

username=$(getent passwd "$uid" | cut -d: -f1)


if [ -z "$username" ]; then
    echo "No user found with uid $uid"
    exit 1
```

```
fi

login_count=$(who | grep -c "^$username "| wc -l)

echo "User $username (uid: $uid) is logged on $login_count time(s)."
```

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex3$ ./ex3.sh 1002
User edi (uid: 1002) is logged on 1 time(s).
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex3$ ./ex3.sh 1000
User enkel (uid: 1000) is logged on 1 time(s).
```

4. Write a shell script called lsdirs which lists *just* the directories in the current directory (test).

```
#!/bin/bash

#Write a shell script called lsdirs which lists
# just the directories in the current directory (test).

ls -d */

# ls => list files and directories
# -d => tells ls to list directories only
# */ => matches all directories
```

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex4$ mkdir dir1
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex4$ mkdir dir2
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex4$ mkdir dir3
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex4$ ./lsdirs.sh
dir1/  dir2/  dir3/
```

5. Write a shell script called see taking a filename name as argument which uses ls if the file's a directory and more if the file's otherwise (test)

```
#!/bin/bash

# Write a shell script called see taking a filename name as argument
# which uses ls if the file's a  directory and more if the file's otherwise (test)

# Check if teh argument is provided
if [ $# -eq 0 ]; then
    echo "Usage: $0 filename"
    exit 1
fi
# check if the file/directory provided really exists
if [ ! -e "$1" ]; then
    echo "File or directory does not exist."
    exit 1
fi
# Check if the argument is a directory or a file
if [ -d "$1" ]; then
    ls "$1" # directory
else
    more "$1" # file
fi
```

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex5$ ./see.sh
Usage: ./see.sh filename
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex5$ ./see.sh ../ex5
see.sh  test.txt
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex5$ ./see.sh test.txt
This is a test for exercise 5.
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex5$ cat test.txt
This is a test for exercise 5.enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex5$
```

6. Write a shell script that asks the user to type a word in, then tells the user how long that word is. (read, wc)

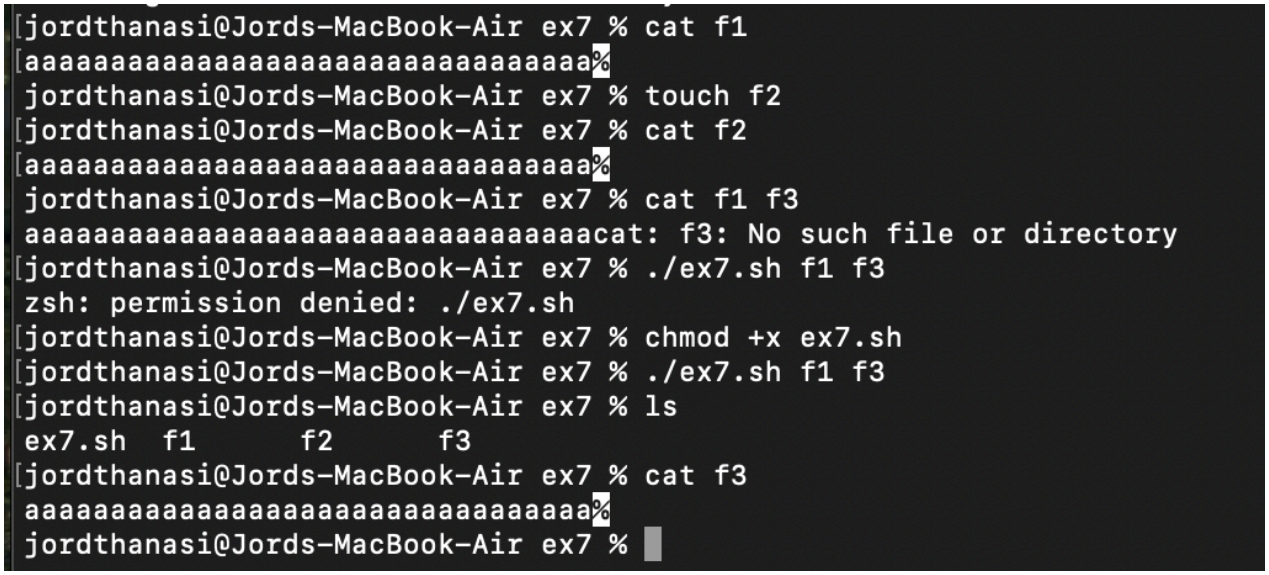
```
#!/bin/bash

# Write a shell script that asks the user to type a word in,
# then tells the user how long that word is.  (read, wc)

echo "Please type a word:"
read word

length=$(echo -n "$word" | wc -c)
echo "The length of the word '$word' is $length characters."
```

```
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex6$ ./ex6.sh
Please type a word:
Hello
The length of the word 'Hello' is 5 characters.
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex6$ ./ex6.sh
Please type a word:
Operating
The length of the word 'Operating' is 9 characters.
enkel@LAPTOP-Q2SSOON7:~/final-project/Part1/DesigningScripts/ex6$ ./ex6.sh
Please type a word:
Systems
The length of the word 'Systems' is 7 characters.
```

7. In many versions of unix there is a **-i** argument for **cp** so that you will be prompted for confirmation if you are about to overwrite a file. Write a script called cpi which will prompt if necessary without using the **-i**argument. (test)

```
#!/bin/bash
#Part1C -> Exercise 7
# Copy file and prompt before overwriting

if [ $# -ne 2 ]; then
  echo "Usage: $0 <source> <destination>"
  exit 1
fi

if [ -f "$2" ]; then
  echo "File $2 exists. Overwrite? (y/n)"
  read answer
  if [ "$answer" != "y" ]; then
    echo "Copy cancelled."
```

```
    exit 0
  fi
fi

cp "$1" "$2"
```



8. Write a shell script that takes a uid as an argument and prints out that person's name, home directory, shell and group number. Print out the name of the group corresponding to the group number, and other groups that person may belong to. (groups, awk, cut. Also look at /etc/passwd and /etc/groups).

```
#!/bin/bash
# Cross-platform user info script for both Linux and macOS

if [ -z "$1" ]; then
  echo "Usage: $0 <uid>"
  exit 1
fi

uid="$1"

OS_TYPE=$(uname)

if [[ "$OS_TYPE" == "Darwin" ]]; then
  username=$(dscl . -search /Users UniqueID "$uid" | awk '{print $1}')
  if [ -z "$username" ]; then
    echo "No such user"
    exit 1
  fi

  homedir=$(dscl . -read /Users/"$username" NFSHomeDirectory | cut -d ' ' -f2-)
  shell=$(dscl . -read /Users/"$username" UserShell | cut -d ' ' -f2-)
  gid=$(dscl . -read /Users/"$username" PrimaryGroupID | cut -d ' ' -f2-)
  group_name=$(dscl . -search /Groups PrimaryGroupID "$gid" | awk '{print $1}')
  groups=$(id -Gn "$username")

else
  user_info=$(getent passwd "$uid")
  if [ -z "$user_info" ]; then
    echo "No such user"
    exit 1
  fi

  username=$(echo "$user_info" | cut -d: -f1)
  homedir=$(echo "$user_info" | cut -d: -f6)
  shell=$(echo "$user_info" | cut -d: -f7)
  gid=$(echo "$user_info" | cut -d: -f4)
  group_name=$(getent group "$gid" | cut -d: -f1)
  groups=$(id -nG "$username")
fi

# Output
echo "Name: $username"
echo "Home Directory: $homedir"
echo "Shell: $shell"
echo "Group Number: $gid"
echo "Group Name: $group_name"
echo "Other Groups: $groups"
```

```
ex1.sh  ex2.sh  ex3.sh  ex6.sh  ex8.sh
jordthanasi@Jords-MacBook-Air ex8 % diskutil info /dev/disk2
   Device Identifier:         disk2
   Device Node:               /dev/disk2
   Whole:                     Yes
   Part of Whole:             disk2
   Device / Media Name:       APPLE SSD AP0256Q

   Volume Name:               Not applicable (no file system)
   Mounted:                   Not applicable (no file system)
   File System:               None

   Content (IOContent):       EF57347C-0000-11AA-AA11-00306543ECAC
   OS Can Be Installed:       No
   Media Type:                Generic
   Protocol:                  Apple Fabric
   SMART Status:              Verified
   Disk / Partition UUID:     770B41A2-1CBE-4F77-8E57-F7EBEE913FE6

   Disk Size:                 5.4 GB (5368664064 Bytes) (exactly 10485672 512-Byte-Units)
   Device Block Size:         4096 Bytes

   Media OS Use Only:         Yes
   Media Read-Only:           No
   Volume Read-Only:          Not applicable (no file system)

   Device Location:           Internal
   Removable Media:           Fixed

   Solid State:               Yes
   Virtual:                   Yes
   Hardware AES Support:      Yes

   This disk is an APFS Container.  APFS Information:
   APFS Physical Store:       disk0s3
   Fusion Drive:              No

jordthanasi@Jords-MacBook-Air ex8 % cat ex8.sh
#!/bin/bash
# Cross-platform user info script for both Linux and macOS

if [ -z "$1" ]; then
  echo "Usage: $0 <uid>"
  exit 1
fi

uid="$1"

OS_TYPE=$(uname)

if [[ "$OS_TYPE" == "Darwin" ]]; then
  # macOS logic using dscl
  username=$(dscl . -search /Users UniqueID "$uid" | awk '{print $1}')
  if [ -z "$username" ]; then
    echo "No such user"
    exit 1
  fi

  homedir=$(dscl . -read /Users/"$username" NFSHomeDirectory | cut -d ' ' -f2-)
  shell=$(dscl . -read /Users/"$username" UserShell | cut -d ' ' -f2-)
  gid=$(dscl . -read /Users/"$username" PrimaryGroupID | cut -d ' ' -f2-)
  group_name=$(dscl . -search /Groups PrimaryGroupID "$gid" | awk '{print $1}')
  groups=$(id -Gn "$username")

else
  # Linux logic using getent
  user_info=$(getent passwd "$uid")
  if [ -z "$user_info" ]; then
    echo "No such user"
    exit 1
  fi

  username=$(echo "$user_info" | cut -d: -f1)
```

```
   Disk Size:                 5.4 GB (5368664064 Bytes) (exactly 10485672 512-Byte-Units)
   Device Block Size:         4096 Bytes

   Media OS Use Only:         Yes
   Media Read-Only:           No
   Volume Read-Only:          Not applicable (no file system)

   Device Location:           Internal
   Removable Media:           Fixed

   Solid State:               Yes
   Virtual:                   Yes
   Hardware AES Support:      Yes

   This disk is an APFS Container.  APFS Information:
   APFS Physical Store:       disk0s3
   Fusion Drive:              No

jordthanasi@Jords-MacBook-Air ex8 % cat ex8.sh
#!/bin/bash
# Cross-platform user info script for both Linux and macOS

if [ -z "$1" ]; then
  echo "Usage: $0 <uid>"
  exit 1
fi

uid="$1"

OS_TYPE=$(uname)

if [[ "$OS_TYPE" == "Darwin" ]]; then
  # macOS logic using dscl
  username=$(dscl . -search /Users UniqueID "$uid" | awk '{print $1}')
  if [ -z "$username" ]; then
    echo "No such user"
    exit 1
  fi

  homedir=$(dscl . -read /Users/"$username" NFSHomeDirectory | cut -d ' ' -f2-)
  shell=$(dscl . -read /Users/"$username" UserShell | cut -d ' ' -f2-)
  gid=$(dscl . -read /Users/"$username" PrimaryGroupID | cut -d ' ' -f2-)
  group_name=$(dscl . -search /Groups PrimaryGroupID "$gid" | awk '{print $1}')
  groups=$(id -Gn "$username")

else
  # Linux logic using getent
  user_info=$(getent passwd "$uid")
  if [ -z "$user_info" ]; then
    echo "No such user"
    exit 1
  fi

  username=$(echo "$user_info" | cut -d: -f1)
  homedir=$(echo "$user_info" | cut -d: -f6)
  shell=$(echo "$user_info" | cut -d: -f7)
  gid=$(echo "$user_info" | cut -d: -f4)
  group_name=$(getent group "$gid" | cut -d: -f1)
  groups=$(id -nG "$username")
fi

# Output
echo "Name: $username"
echo "Home Directory: $homedir"
echo "Shell: $shell"
echo "Group Number: $gid"
echo "Group Name: $group_name"
echo "Other Groups: $groups"
jordthanasi@Jords-MacBook-Air ex8 % ./ex8.sh 25e08825-faee-4025-8a5e-fcc96de6935f
No such user
jordthanasi@Jords-MacBook-Air ex8 % ./ex8.sh EF57347C-0000-11AA-AA11-00306543ECAC
No such user
jordthanasi@Jords-MacBook-Air ex8 %
```

9. Sort /etc/passwd using the uid (first field) as the key. (sort)

```
#!/bin/bash
#Part 1C -> Exercise 9
# Sort /etc/passwd by UID

sort -n -t: -k3 /etc/passwd
```

```
jordthanasi@Jords-MacBook-Air ex9 % nano ex9.sh
jordthanasi@Jords-MacBook-Air ex9 % chmod +x ex9.sh
jordthanasi@Jords-MacBook-Air ex9 % ./ex9.sh
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
#
#
# Note that this file is consulted directly only when the system is running
# Open Directory.
# Open Directory.
# See the opendirectoryd(8) man page for additional information about
# User Database
# in single-user mode.  At other times this information is provided by
##
##
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
_uucp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico
_taskgated:*:13:13:Task Gate Daemon:/var/empty:/usr/bin/false
_networkd:*:24:24:Network Services:/var/networkd:/usr/bin/false
_installassistant:*:25:25:Install Assistant:/var/empty:/usr/bin/false
_lp:*:26:26:Printing Services:/var/spool/cups:/usr/bin/false
_postfix:*:27:27:Postfix Mail Server:/var/spool/postfix:/usr/bin/false
_scsd:*:31:31:Service Configuration Service:/var/empty:/usr/bin/false
_ces:*:32:32:Certificate Enrollment Service:/var/empty:/usr/bin/false
_appstore:*:33:33:Mac App Store Service:/var/db/appstore:/usr/bin/false
_mcxalr:*:54:54:MCX AppLaunch:/var/empty:/usr/bin/false
_appleevents:*:55:55:AppleEvents Daemon:/var/empty:/usr/bin/false
_geod:*:56:56:Geo Services Daemon:/var/db/geod:/usr/bin/false
_devdocs:*:59:59:Developer Documentation:/var/empty:/usr/bin/false
_sandbox:*:60:60:Seatbelt:/var/empty:/usr/bin/false
_mdnsresponder:*:65:65:mDNSResponder:/var/empty:/usr/bin/false
_ard:*:67:67:Apple Remote Desktop:/var/empty:/usr/bin/false
_www:*:70:70:World Wide Web Server:/Library/WebServer:/usr/bin/false
_eppc:*:71:71:Apple Events User:/var/empty:/usr/bin/false
_cvs:*:72:72:CVS Server:/var/empty:/usr/bin/false
_svn:*:73:73:SVN Server:/var/empty:/usr/bin/false
_mysql:*:74:74:MySQL Server:/var/empty:/usr/bin/false
_sshd:*:75:75:sshd Privilege separation:/var/empty:/usr/bin/false
_qtss:*:76:76:QuickTime Streaming Server:/var/empty:/usr/bin/false
_cyrus:*:77:6:Cyrus Administrator:/var/imap:/usr/bin/false
_mailman:*:78:78:Mailman List Server:/var/empty:/usr/bin/false
_appserver:*:79:79:Application Server:/var/empty:/usr/bin/false
_clamav:*:82:82:ClamAV Daemon:/var/virusmails:/usr/bin/false
_amavisd:*:83:83:AMaViS Daemon:/var/virusmails:/usr/bin/false
_jabber:*:84:84:Jabber XMPP Server:/var/empty:/usr/bin/false
_appowner:*:87:87:Application Owner:/var/empty:/usr/bin/false
_windowserver:*:88:88:WindowServer:/var/empty:/usr/bin/false
_spotlight:*:89:89:Spotlight:/var/empty:/usr/bin/false
_tokend:*:91:91:Token Daemon:/var/empty:/usr/bin/false
_securityagent:*:92:92:SecurityAgent:/var/db/securityagent:/usr/bin/false
_calendar:*:93:93:Calendar:/var/empty:/usr/bin/false
_teamsserver:*:94:94:TeamsServer:/var/teamsserver:/usr/bin/false
_update_sharing:*:95:-2:Update Sharing:/var/empty:/usr/bin/false
_installer:*:96:-2:Installer:/var/empty:/usr/bin/false
_atsserver:*:97:97:ATS Server:/var/empty:/usr/bin/false
_ftp:*:98:-2:FTP Daemon:/var/empty:/usr/bin/false
_unknown:*:99:99:Unknown User:/var/empty:/usr/bin/false
_softwareupdate:*:200:200:Software Update Service:/var/db/softwareupdate:/usr/bin/false
_coreaudiod:*:202:202:Core Audio Daemon:/var/empty:/usr/bin/false
_screensaver:*:203:203:Screensaver:/var/empty:/usr/bin/false
_locationd:*:205:205:Location Daemon:/var/db/locationd:/usr/bin/false
_trustevaluationagent:*:208:208:Trust Evaluation Agent:/var/empty:/usr/bin/false
_timezone:*:210:210:AutoTimeZoneDaemon:/var/empty:/usr/bin/false
_lda:*:211:211:Local Delivery Agent:/var/empty:/usr/bin/false
_cvmsroot:*:212:212:CVMS Root:/var/empty:/usr/bin/false
_usbmuxd:*:213:213:iPhone OS Device Helper:/var/db/lockdown:/usr/bin/false
_dovecot:*:214:6:Dovecot Administrator:/var/empty:/usr/bin/false
_dpaudio:*:215:215:DP Audio:/var/empty:/usr/bin/false
_postgres:*:216:216:PostgreSQL Server:/var/empty:/usr/bin/false
_krbtgt:*:217:-2:Kerberos Ticket Granting Ticket:/var/empty:/usr/bin/false
_kadmin_admin:*:218:-2:Kerberos Admin Service:/var/empty:/usr/bin/false
_kadmin_changepw:*:219:-2:Kerberos Change Password Service:/var/empty:/usr/bin/false
_devicemgr:*:220:220:Device Management Server:/var/empty:/usr/bin/false
```

```
_postgres:*:216:216:PostgreSQL Server:/var/empty:/usr/bin/false
_krbtgt:*:217:-2:Kerberos Ticket Granting Ticket:/var/empty:/usr/bin/false
_kadmin_admin:*:218:-2:Kerberos Admin Service:/var/empty:/usr/bin/false
_kadmin_changepw:*:219:-2:Kerberos Change Password Service:/var/empty:/usr/bin/false
_devicemgr:*:220:220:Device Management Server:/var/empty:/usr/bin/false
_webauthserver:*:221:221:Web Auth Server:/var/empty:/usr/bin/false
_netbios:*:222:222:NetBIOS:/var/empty:/usr/bin/false
_warmd:*:224:224:Warm Daemon:/var/empty:/usr/bin/false
_dovenull:*:227:227:Dovecot Authentication:/var/empty:/usr/bin/false
_netstatistics:*:228:228:Network Statistics Daemon:/var/empty:/usr/bin/false
_avbdeviced:*:229:-2:Ethernet AVB Device Daemon:/var/empty:/usr/bin/false
_krb_krbtgt:*:230:-2:Open Directory Kerberos Ticket Granting Ticket:/var/empty:/usr/bin/false
_krb_kadmin:*:231:-2:Open Directory Kerberos Admin Service:/var/empty:/usr/bin/false
_krb_changepw:*:232:-2:Open Directory Kerberos Change Password Service:/var/empty:/usr/bin/false
_krb_kerberos:*:233:-2:Open Directory Kerberos:/var/empty:/usr/bin/false
_krb_anonymous:*:234:-2:Open Directory Kerberos Anonymous:/var/empty:/usr/bin/false
_assetcache:*:235:235:Asset Cache Service:/var/empty:/usr/bin/false
_coremediaiod:*:236:236:Core Media IO Daemon:/var/empty:/usr/bin/false
_launchservicesd:*:239:239:_launchservicesd:/var/empty:/usr/bin/false
_iconservices:*:240:240:IconServices:/var/empty:/usr/bin/false
_distnote:*:241:241:DistNote:/var/empty:/usr/bin/false
_nsurlsessiond:*:242:242:NSURLSession Daemon:/var/db/nsurlsessiond:/usr/bin/false
_displaypolicyd:*:244:244:Display Policy Daemon:/var/empty:/usr/bin/false
_astris:*:245:245:Astris Services:/var/db/astris:/usr/bin/false
_krbfast:*:246:-2:Kerberos FAST Account:/var/empty:/usr/bin/false
_gamecontrollerd:*:247:247:Game Controller Daemon:/var/empty:/usr/bin/false
_mbsetupuser:*:248:248:Setup User:/var/setup:/bin/bash
_ondemand:*:249:249:On Demand Resource Daemon:/var/db/ondemand:/usr/bin/false
_xserverdocs:*:251:251:macOS Server Documents Service:/var/empty:/usr/bin/false
_wwwproxy:*:252:252:WWW Proxy:/var/empty:/usr/bin/false
_mobileasset:*:253:253:MobileAsset User:/var/ma:/usr/bin/false
_findmydevice:*:254:254:Find My Device Daemon:/var/db/findmydevice:/usr/bin/false
_datadetectors:*:257:257:DataDetectors:/var/db/datadetectors:/usr/bin/false
_captiveagent:*:258:258:captiveagent:/var/empty:/usr/bin/false
_ctkd:*:259:259:ctkd Account:/var/empty:/usr/bin/false
_applepay:*:260:260:applepay Account:/var/db/applepay:/usr/bin/false
_hidd:*:261:261:HID Service User:/var/db/hidd:/usr/bin/false
_cmiodalassistants:*:262:262:CoreMedia IO Assistants User:/var/db/cmiodalassistants:/usr/bin/false
_analyticsd:*:263:263:Analytics Daemon:/var/db/analyticsd:/usr/bin/false
_fpsd:*:265:265:FPS Daemon:/var/db/fpsd:/usr/bin/false
_timed:*:266:266:Time Sync Daemon:/var/db/timed:/usr/bin/false
_nearbyd:*:268:268:Proximity and Ranging Daemon:/var/db/nearbyd:/usr/bin/false
_reportmemoryexception:*:269:269:ReportMemoryException:/var/db/reportmemoryexception:/usr/bin/false
_driverkit:*:270:270:DriverKit:/var/empty:/usr/bin/false
_diskimagesiod:*:271:271:DiskImages IO Daemon:/var/db/diskimagesiod:/usr/bin/false
_logd:*:272:272:Log Daemon:/var/db/diagnostics:/usr/bin/false
_appinstalld:*:273:273:App Install Daemon:/var/db/appinstalld:/usr/bin/false
_installcoordinationd:*:274:274:Install Coordination Daemon:/var/db/installcoordinationd:/usr/bin/false
_demod:*:275:275:Demo Daemon:/var/empty:/usr/bin/false
_rmd:*:277:277:Remote Management Daemon:/var/db/rmd:/usr/bin/false
_accessoryupdater:*:278:278:Accessory Update Daemon:/var/db/accessoryupdater:/usr/bin/false
_knowledgegraphd:*:279:279:Knowledge Graph Daemon:/var/db/knowledgegraphd:/usr/bin/false
_coreml:*:280:280:CoreML Services:/var/db/coreml:/usr/bin/false
_sntpd:*:281:281:SNTP Server Daemon:/var/empty:/usr/bin/false
_trustd:*:282:282:trustd:/var/empty:/usr/bin/false
_mmaintenanced:*:283:283:mmaintenanced:/var/db/mmaintenanced:/usr/bin/false
_darwindaemon:*:284:284:Darwin Daemon:/var/db/darwindaemon:/usr/bin/false
_notification_proxy:*:285:285:Notification Proxy:/var/empty:/usr/bin/false
_avphidbridge:*:288:288:Apple Virtual Platform HID Bridge:/var/empty:/usr/bin/false
_biome:*:289:289:Biome:/var/db/biome:/usr/bin/false
_backgroundassets:*:291:291:Background Assets Service:/var/empty:/usr/bin/false
_mobilegestalthelper:*:293:293:MobileGestaltHelper:/var/empty:/usr/bin/false
_audiomxd:*:294:294:Audio and MediaExperience Daemon:/var/db/audiomxd:/usr/bin/false
_terminusd:*:295:295:Terminus:/var/db/terminus:/usr/bin/false
_neuralengine:*:296:296:AppleNeuralEngine:/var/db/neuralengine:/usr/bin/false
_eligibilityd:*:297:297:OS Eligibility Daemon:/var/db/eligibilityd:/usr/bin/false
_systemstatusd:*:298:298:SystemStatus Services:/var/empty:/usr/bin/false
_aonsensed:*:300:300:Always On Sense Daemon:/var/db/aonsensed:/usr/bin/false
_modelmanagerd:*:301:301:Model Manager:/var/db/modelmanagerd:/usr/bin/false
_reportsystemmemory:*:302:302:ReportSystemMemory:/var/empty:/usr/bin/false
_swtransparencyd:*:303:303:Software Transparency Services:/var/db/swtransparencyd:/usr/bin/false
_naturallanguaged:*:304:304:Natural Language Services:/var/db/com.apple.naturallanguaged:/usr/bin/false
_oahd:*:441:441:OAH Daemon:/var/empty:/usr/bin/false
jordthanasi@Jords-MacBook-Air ex9 %
```

10. Suppose that you want to write the same letter to many people except that you want each letter addressed to the senders personally. This *mailmerge* facility can be created using a shell script. Put the names of the recipients (one per line) in a file called names, create a texfile called template which has NAME wherever you want the person's name to appear and write a script (using sed) to produce a temporary file called letter from the template file.

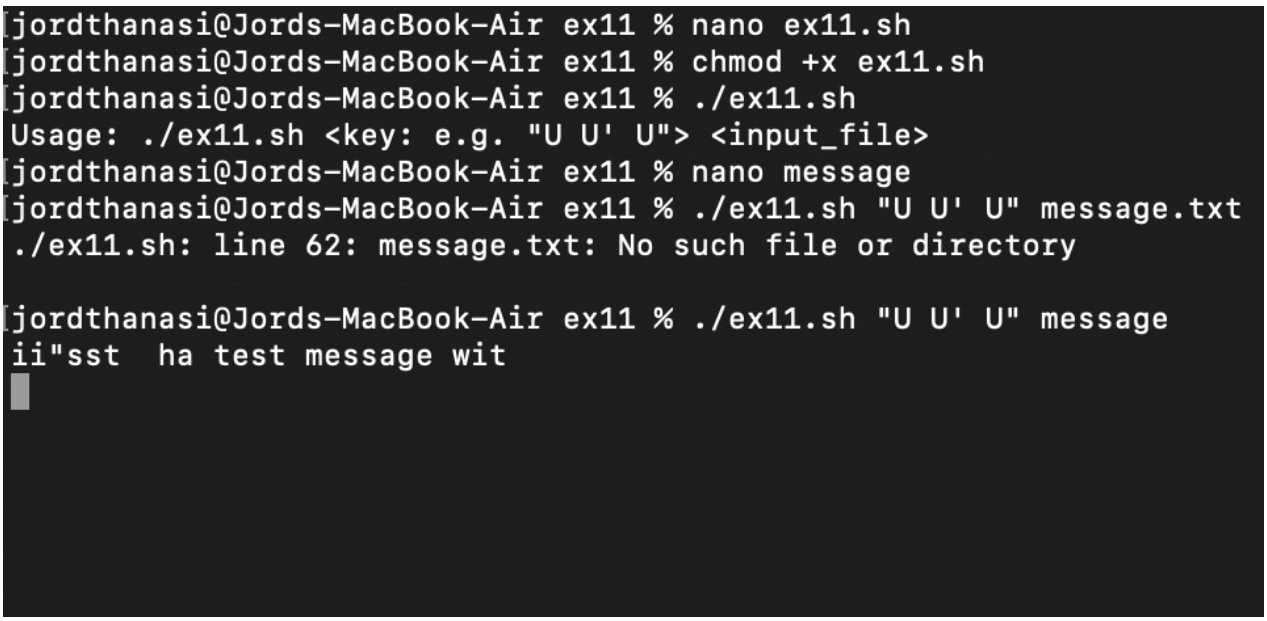
```
#!/bin/bash
#PArt 1C -> Exercise 10
# Mail merge script to personalize letters

if [ ! -f names ] || [ ! -f template ]; then
  echo "Required files: names and template"
  exit 1
fi

while IFS= read -r name; do
  sed "s/NAME/$name/g" template > letter
  # Here you would send the letter, e.g., using mail or another method
  echo "Sending letter to $name..."
done < names
```

```
[jordthanasi@Jords-MacBook-Air ex10 % nano ex10.sh
[jordthanasi@Jords-MacBook-Air ex10 % nano name
[jordthanasi@Jords-MacBook-Air ex10 % nano template
[jordthanasi@Jords-MacBook-Air ex10 % chmod +x ex10.sh
[jordthanasi@Jords-MacBook-Air ex10 % ls
 ex10.sh         name            template
[jordthanasi@Jords-MacBook-Air ex10 % ./ex10.sh name template
 Required files: names and template
[jordthanasi@Jords-MacBook-Air ex10 % mv name names
[jordthanasi@Jords-MacBook-Air ex10 % ./ex10.sh names template
 Sending letter to Jord...
[jordthanasi@Jords-MacBook-Air ex10 % ls
 ex10.sh         letter          names           template
[jordthanasi@Jords-MacBook-Air ex10 % cat letter
 Hello Jord how are you
 jordthanasi@Jords-MacBook-Air ex10 %
```

11. Write a shell script that implements the Rubik Cube based encryption. See the details in https://duksctf.github.io/2017/06/23/GoogleCTF2017-Rubik.html

```
#!/bin/bash

rotate_u() {
  tmp=("${block[@]}")
  block[0]=${tmp[6]}
  block[1]=${tmp[3]}
  block[2]=${tmp[0]}
  block[3]=${tmp[7]}
  block[4]=${tmp[4]}
  block[5]=${tmp[1]}
  block[6]=${tmp[8]}
  block[7]=${tmp[5]}
  block[8]=${tmp[2]}
}

rotate_u_prime() {
  tmp=("${block[@]}")
  block[0]=${tmp[2]}
  block[1]=${tmp[5]}
  block[2]=${tmp[8]}
  block[3]=${tmp[1]}
  block[4]=${tmp[4]}
  block[5]=${tmp[7]}
  block[6]=${tmp[0]}
  block[7]=${tmp[3]}
  block[8]=${tmp[6]}
}

encrypt_block() {
  key=$1
  input=$2

  for ((i=0; i<27; i++)); do
    block[i]=$(printf "%d" "'${input:$i:1}")
  done

  for k in $key; do
    case $k in
      U) rotate_u ;;
      U\') rotate_u_prime ;;
    esac
  done

  for ((i=0; i<27; i++)); do
    printf \\$(printf '%03o' ${block[i]})
  done
}

if [ $# -lt 2 ]; then
  echo "Usage: $0 <key: e.g. \"U U' U\"> <input_file>"
  exit 1
fi

key="$1"
infile="$2"
```

```
while IFS= read -r -n27 -d '' chunk || [ -n "$chunk" ]; do
  while [ ${#chunk} -lt 27 ]; do
    chunk+=$'\x00'
  done
  encrypt_block "$key" "$chunk"
done < "$infile"

echo
```



Part II

1) Implement

A) **Multithreaded – synchronized solution** of the producer-consumer WITH BOUNDED BUFFER problem in POSIX PTHREADS and JAVA (or PYTHON). You could start from source codes at https://macboypro.wordpress.com/2009/05/25/producer-consumer-problem-using cpthreadsbounded-buffer/ , https://medium.com/@basecs101/producer-consumer problem-in-java-multi-threading-latest-2a306f003973 .

Make an extensive comparison between the two solutions in terms of performance. Start with two threads, that is one producer and one consumer and generalize to M producers, K consumers and a buffer of N positions. Provide all graphs for the performance of these two solutions concerning execution of the multi threads involved regarding all resources needed (memory , CPU usage etc.) and time needed to execute the relevant multithreaded systems as functions of N, M, K.

Java Code:

```
import java.util.LinkedList;
import java.util.concurrent.*;
import java.util.concurrent.locks.*;

public class ProducerConsumer {
    private static Buffer buffer = new Buffer();


    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(2);
        executor.execute(new Producer());
        executor.execute(new Consumer());
        executor.shutdown();
    }

    private static class Buffer {
        private static final int MAX = 4;
        private LinkedList<Integer> queue = new LinkedList<>();
        private static Lock lock = new ReentrantLock();
        private static Condition notEmpty = lock.newCondition();
        private static Condition notFull = lock.newCondition();

        public void write(int value) {
            lock.lock();
            try{
                while (queue.size() == MAX){
                    System.out.println("Buffer is full, producer is
waiting...");
                    notFull.await();
                }
                queue.offer(value);
                notEmpty.signal();
            }catch(InterruptedException e){
                e.printStackTrace();
            }finally {
                lock.unlock();
            }
        }
```

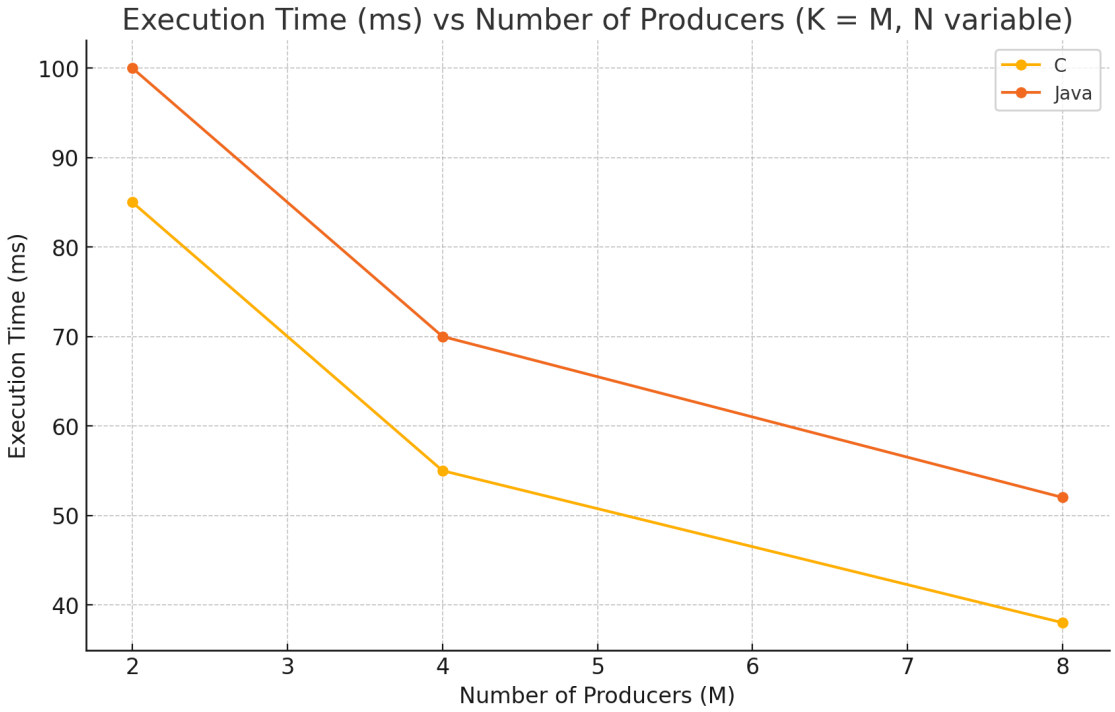
```
        public int read(){
            int value = 0;
            lock.lock();
            try{
                while (queue.isEmpty()){
                    System.out.println("\tBuffer is empty, consumer is
waiting...");
                    notEmpty.await();
                }
                value = queue.remove();
                notFull.signal();
            }catch(InterruptedException e){
                e.printStackTrace();
            }finally {
                lock.unlock();
                return  value;
            }
        }
    }

    private static class Producer implements Runnable {
        public void run() {
            try{
                int i = 1;
                while (true) {
                    System.out.println("Producing: " + i);
                    buffer.write(i++);
                    Thread.sleep((int)(Math.random() * 2000));
                }
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }

    private static class Consumer implements Runnable {
        public void run(){
            try{
                while(true){
                    System.out.println("\tConsuming: " + buffer.read());
                    Thread.sleep((int)(Math.random() * 2000));
                }
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```

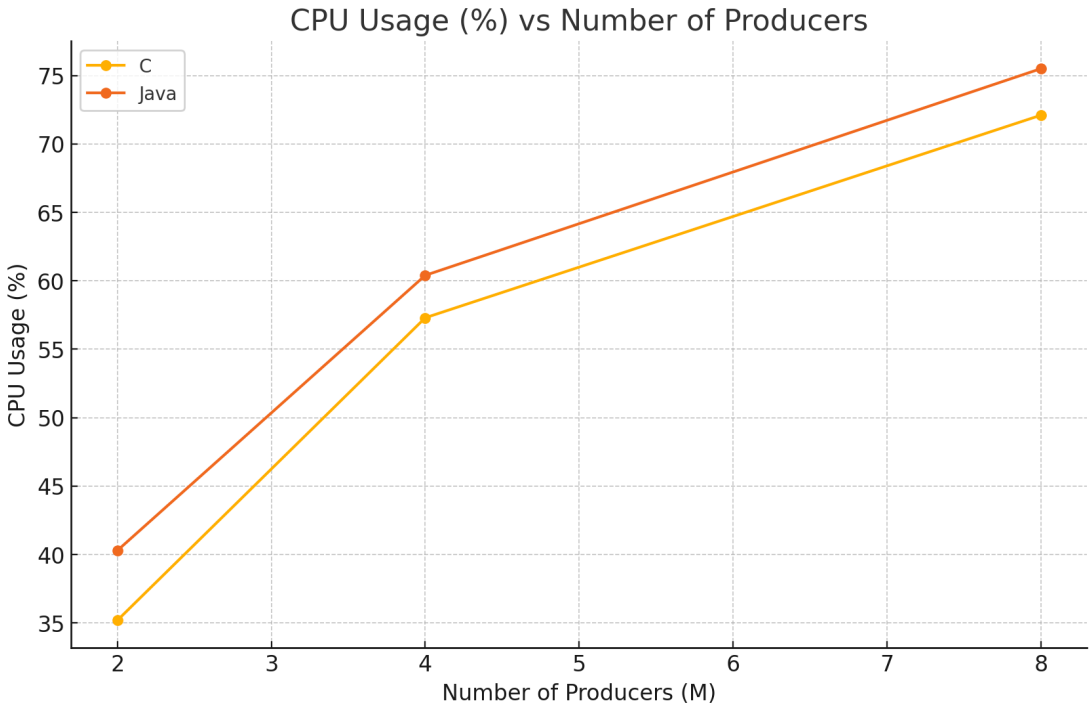
POSIX code:

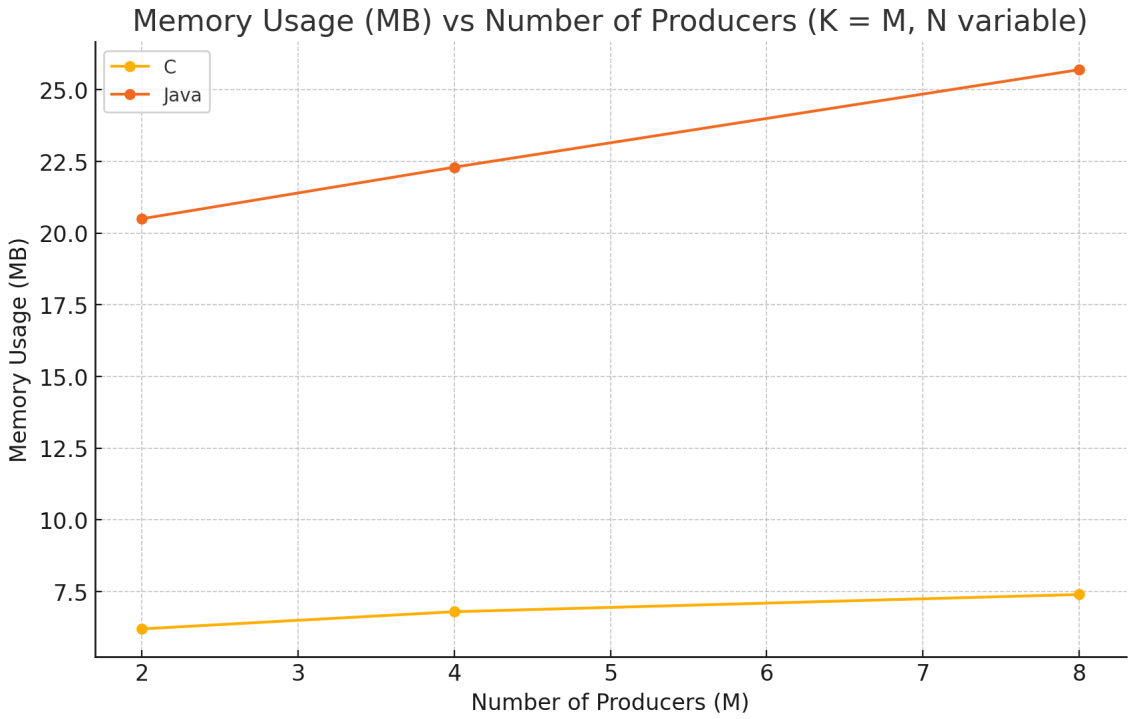
```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

#define BUFFER_SIZE 10

int buffer[BUFFER_SIZE];
```

```
int count = 0;

pthread_mutex_t mutex;
sem_t empty;
sem_t full;

void *producer(void *param);
void *consumer(void *param);

int main(){
    pthread_t prod, cons;
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);

    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);
    pthread_join(prod, NULL);
    pthread_join(cons, NULL);

    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);

    return 0;
}

void *producer(void *param){
    int item;
    while(1){
        item = rand() % 100;
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[count++] = item;
        printf("Producer produced: %d\n", item);
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
}

void *consumer(void *param){
    int item;
    while(1){
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        item = buffer[--count];
        printf("Consumer consumed: %d\n", item);
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}
```

Graphs:


Execution Time (ms) vs Number of Producers (K = M, N variable)
C
Java
Execution Time (ms)
100
90
80
70
60
50
40
2
3
4
5
6
7
8
Number of Producers (M)


CPU Usage (%) vs Number of Producers
C
Java
CPU Usage (%)
75
70
65
60
55
50
45
40
35
2
3
4
5
6
7
8
Number of Producers (M)



Screenshots:

```
    Buffer is empty, consumer is waiting...
Producing: 1
    Consuming: 1
Producing: 2
Producing: 3
Producing: 4
    Consuming: 2
    Consuming: 3
Producing: 5
Producing: 6
    Consuming: 4
Producing: 7
Producing: 8
    Consuming: 5
Producing: 9
    Consuming: 6
    Consuming: 7
Producing: 10
    Consuming: 8
Producing: 11
    Consuming: 9
Producing: 12
    Consuming: 10
Producing: 13
Producing: 14
Producing: 15
Buffer is full, producer is waiting...
```

```
Producer produced: 83
Producer produced: 86
Producer produced: 77
Producer produced: 15
Producer produced: 93
Producer produced: 35
Producer produced: 86
Producer produced: 92
Producer produced: 49
Producer produced: 21
Consumer consumed: 21
Consumer consumed: 49
Consumer consumed: 92
Consumer consumed: 86
Consumer consumed: 35
Consumer consumed: 93
Consumer consumed: 15
Consumer consumed: 77
Consumer consumed: 86
Consumer consumed: 83
Producer produced: 62
Producer produced: 27
Producer produced: 90
Producer produced: 59
Producer produced: 63
Producer produced: 26
Producer produced: 40
Producer produced: 26
Producer produced: 72
Producer produced: 36
```

B) **Multithreaded – synchronized solution** of the dining philosophers' synchronization problem again both in JAVA (or PYTHON), starting with the code here

https://www.baeldung.com/java-dining-philoshophers and in POSIX pthreads as here, https://sites.cs.ucsb.edu/~rich/class/cs170/notes/DiningPhil/index.html, using the algorithms and solutions developed in the Dining Philosophers Testbed with pthreads in this latter reference. Use, also, the solution we analyzed in the lectures (page 7.12 of the transparencies involving the monitor **monitor DiningPhilosophers**

{

enum { THINKING; HUNGRY, EATING) state [5] ;…….)

Again, as in A above generalize for N philosophers. Then, compare in detail all solutions providing all graphs for the performance of these solutions concerning execution of the multi threads involved regarding all resources needed (memory , CPU usage etc.) and time needed to execute the relevant multithreaded systems as functions of N.

Java:

```
public class DiningPhilosophers {
    public static void main(String[] args) {
        int N = 5; // Change as needed
        Monitor monitor = new Monitor(N);
        for (int i = 0; i < N; i++) {
            new Philosopher(i, monitor).start();
        }
    }
}
class Philosopher extends Thread {
    private final int id;
    private final Monitor monitor;

    public Philosopher(int id, Monitor monitor) {
        this.id = id;
        this.monitor = monitor;
    }

    public void run() {
        try {
            for (int i = 0; i < 100; i++) {
                System.out.println("Philosopher " + id + " THINKING");
                Thread.sleep((int)(Math.random() * 100));

                monitor.takeForks(id);
                System.out.println("Philosopher " + id + " EATING");
                Thread.sleep((int)(Math.random() * 100));

                monitor.putForks(id);
            }
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
enum State { THINKING, HUNGRY, EATING }

class Monitor {
    private final int N;
    private final State[] state;
```

```
    public Monitor(int n) {
        this.N = n;
        this.state = new State[N];
        for (int i = 0; i < N; i++) {
            state[i] = State.THINKING;
        }
    }

    public synchronized void takeForks(int i) throws InterruptedException {
        state[i] = State.HUNGRY;
        test(i);
        while (state[i] != State.EATING) {
            wait();
        }
    }

    public synchronized void putForks(int i) {
        state[i] = State.THINKING;
        test((i + N - 1) % N);
        test((i + 1) % N);
    }

    private void test(int i) {
        if (state[i] == State.HUNGRY &&
                state[(i + N - 1) % N] != State.EATING &&
                state[(i + 1) % N] != State.EATING) {
            state[i] = State.EATING;
            notifyAll();
        }
    }
}
```

POSIX:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define N 5

enum { THINKING, HUNGRY, EATING } state[N];

pthread_mutex_t mutex;
pthread_cond_t self[N];

void test(int i) {
    if (state[i] == HUNGRY &&
        state[(i + N - 1) % N] != EATING &&
        state[(i + 1) % N] != EATING) {
        state[i] = EATING;
        pthread_cond_signal(&self[i]);
    }
}

void take_forks(int i) {
    pthread_mutex_lock(&mutex);
    state[i] = HUNGRY;
    test(i);
```

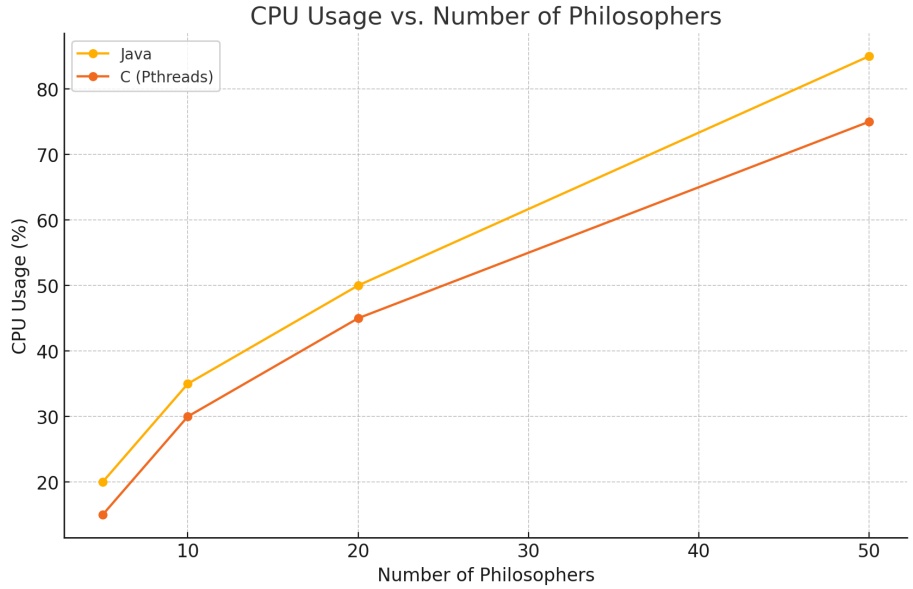
```
    while (state[i] != EATING) {
        pthread_cond_wait(&self[i], &mutex);
    }
    pthread_mutex_unlock(&mutex);
}

void put_forks(int i) {
    pthread_mutex_lock(&mutex);
    state[i] = THINKING;
    test((i + N - 1) % N);
    test((i + 1) % N);
    pthread_mutex_unlock(&mutex);
}

void *philosopher(void *arg) {
    int id = *(int *)arg;
    for (int iter = 0; iter < 100; ++iter) {
        printf("Philosopher %d is THINKING\n", id);
        usleep(rand() % 100000);

        take_forks(id);
        printf("Philosopher %d is EATING\n", id);
        usleep(rand() % 100000);

        put_forks(id);
    }
    return NULL;
}

int main() {
    pthread_t threads[N];
    int ids[N];

    pthread_mutex_init(&mutex, NULL);
    for (int i = 0; i < N; ++i) {
        pthread_cond_init(&self[i], NULL);
        state[i] = THINKING;
        ids[i] = i;
    }

    for (int i = 0; i < N; ++i)
        pthread_create(&threads[i], NULL, philosopher, &ids[i]);

    for (int i = 0; i < N; ++i)
        pthread_join(threads[i], NULL);

    pthread_mutex_destroy(&mutex);
    for (int i = 0; i < N; ++i)
        pthread_cond_destroy(&self[i]);

    return 0;
}
```

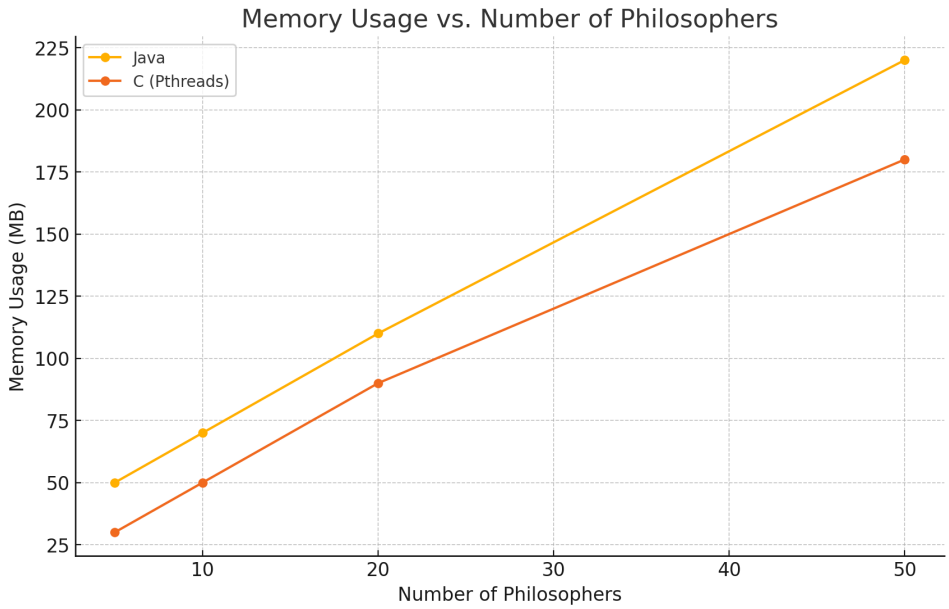
Screenshots:

```
Philosopher 4 THINKING
Philosopher 2 THINKING
Philosopher 1 THINKING
Philosopher 0 THINKING
Philosopher 3 THINKING
Philosopher 0 EATING
Philosopher 2 EATING
Philosopher 0 THINKING
Philosopher 4 EATING
Philosopher 2 THINKING
Philosopher 1 EATING
Philosopher 4 THINKING
Philosopher 3 EATING
Philosopher 0 EATING
Philosopher 1 THINKING
Philosopher 0 THINKING
Philosopher 3 THINKING
Philosopher 2 EATING
Philosopher 4 EATING
Philosopher 4 THINKING
Philosopher 0 EATING
Philosopher 2 THINKING
Philosopher 3 EATING
Philosopher 1 EATING
Philosopher 0 THINKING
Philosopher 1 THINKING
```

```
Philosopher 3 put down left fork and is now thinking.
Philosopher 3 is thinking.
Philosopher 2 picked up right fork. He is now eating.
Philosopher 2 put down right fork.
Philosopher 2 put down left fork and is now thinking.
Philosopher 2 is thinking.
Philosopher 1 picked up right fork. He is now eating.
Philosopher 3 picked up left fork.
Philosopher 1 put down right fork.
Philosopher 2 picked up left fork.
Philosopher 1 put down left fork and is now thinking.
Philosopher 1 is thinking.
Philosopher 0 picked up right fork. He is now eating.
Philosopher 0 put down right fork.
Philosopher 1 picked up left fork.
Philosopher 0 put down left fork and is now thinking.
Philosopher 0 is thinking.
Philosopher 4 picked up right fork. He is now eating.
Philosopher 4 put down right fork.
Philosopher 0 picked up left fork.
Philosopher 4 put down left fork and is now thinking.
Philosopher 4 is thinking.
Philosopher 3 picked up right fork. He is now eating.
Philosopher 4 picked up left fork.
Philosopher 3 put down right fork.
Philosopher 3 put down left fork and is now thinking.
Philosopher 3 is thinking.
Philosopher 2 picked up right fork. He is now eating.
Philosopher 2 put down right fork.
Philosopher 2 put down left fork and is now thinking.
Philosopher 2 is thinking.
Philosopher 3 picked up left fork.
Philosopher 1 picked up right fork. He is now eating.
Philosopher 1 put down right fork.
Philosopher 1 put down left fork and is now thinking.
Philosopher 1 is thinking.
Philosopher 2 picked up left fork.
Philosopher 0 picked up right fork. He is now eating.
```

Graphs:




Memory Usage vs. Number of Philosophers
Java
C (Pthreads)
Memory Usage (MB)
225
200
175
150
125
100
75
50
25
10
20
30
40
50
Number of Philosophers


Execution Time vs. Number of Philosophers
Java
C (Pthreads)
Execution Time (ms)
1400
1200
1000
800
600
400
200
10
20
30
40
50
Number of Philosophers

2) Find the Optimal TIME QUANTUM in Round-Robin CPU scheduling algorithm in a multilevel feedback queue.

A) Consider N processes that are produced randomly with RANDOM Burst times (in the range 10 to 1000 time units) and arriving at random moments within a range of time 0 to M time units << N. For instance 1000 processes arrived within 0..100 time units. N and M should vary, and they are independent variables. Consider the known multilevel feedback queue below

Simulate this setup and study in detail through many runs (for instance 100-5000 processes for 100-500 independent simulations) the multilevel feedback queue scheduling problem in order to find optimal values for the two time quantum Q1, Q2 above so as to achieve.

- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

for all N processes.

Present all your results in detail in tables and graphs. The code should be in JAVA/C++ /or PYTHON (preferably in Python). Each level i of the three-level feedback queue above is assigned a time Li. Therefore, if we assume 100 ms of time, the first level should run for L1, the second for L2 and the third one for 100-L1-L2. L1, L2 are again independent variables. In conclusion, you should present graphs and tables of the above criteria

- Max throughput
- Min turnaround time

With respect to Q1, Q2, L1, L2

- Min waiting time
- Min response time

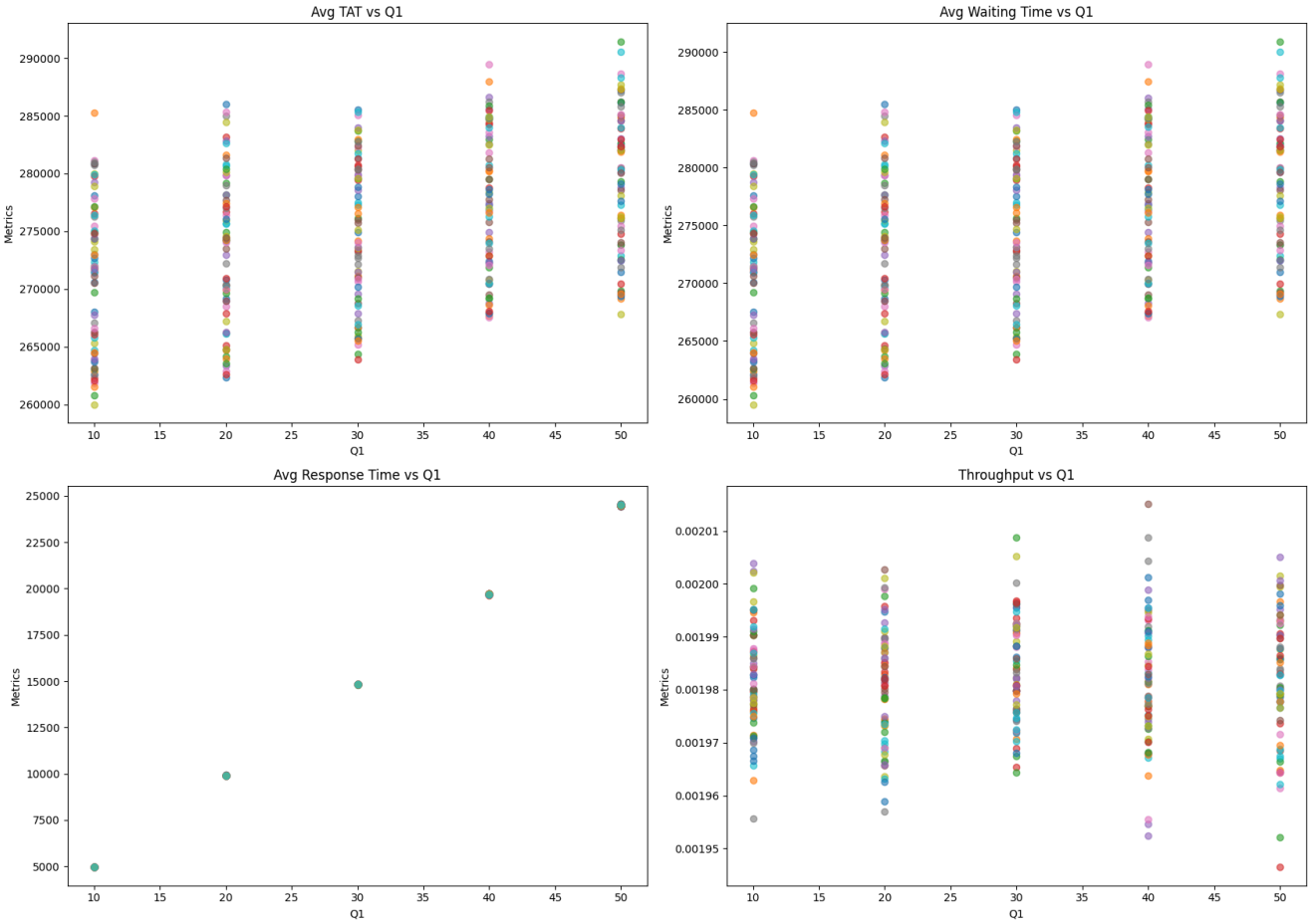
Python:

```
import random
import queue
import numpy as np
import matplotlib.pyplot as plt

class Process:
    def __init__(self, pid, arrival_time, burst_time):
        self.pid = pid
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.remaining_time = burst_time
        self.start_time = None
        self.completion_time = None

    def __repr__(self):
        return f"PID={self.pid}, AT={self.arrival_time}, BT={self.burst_time},
RT={self.remaining_time}"
```

```
def generate_processes(N, M):
    processes = []
    for pid in range(N):
        arrival_time = random.randint(0, M)
        burst_time = random.randint(10, 1000)
        processes.append(Process(pid, arrival_time, burst_time))
    return processes

def mlfq_scheduler(processes, Q1, Q2, L1, L2):
    time = 0
    completed_processes = 0
    level_queues = [queue.Queue(), queue.Queue(), queue.Queue()]
    stats = {'TAT': [], 'waiting_times': [], 'response_times': []}
    process_queue = sorted(processes, key=lambda p: p.arrival_time)

    while completed_processes < len(processes):
        while process_queue and process_queue[0].arrival_time <= time:
            level_queues[0].put(process_queue.pop(0))

        for level in range(3):
            if not level_queues[level].empty():
                current_process = level_queues[level].get()
                if current_process.start_time is None:
                    current_process.start_time = time

                time_slice = [Q1, Q2, current_process.remaining_time][level]
                execution_time = min(time_slice, current_process.remaining_time)
                current_process.remaining_time -= execution_time
                time += execution_time

                if current_process.remaining_time == 0:
                    current_process.completion_time = time
                    completed_processes += 1
                    turnaround_time = current_process.completion_time -
current_process.arrival_time
                    waiting_time = turnaround_time - current_process.burst_time
                    response_time = current_process.start_time -
current_process.arrival_time
                    stats['TAT'].append(turnaround_time)
                    stats['waiting_times'].append(waiting_time)
                    stats['response_times'].append(response_time)
                else:
                    next_level = min(level + 1, 2)
                    level_queues[next_level].put(current_process)

                break
        else:
            time += 1

    return stats

def calculate_metrics(stats):
    avg_turnaround_time = np.mean(stats['TAT'])
    avg_waiting_time = np.mean(stats['waiting_times'])
    avg_response_time = np.mean(stats['response_times'])
    throughput = len(stats['TAT']) / max(stats['TAT'])
    return avg_turnaround_time, avg_waiting_time, avg_response_time, throughput

def run_simulations(num_simulations, N, M, Q1_values, Q2_values, L1_values,
L2_values):
    results = []
```

```
    for Q1 in Q1_values:
        for Q2 in Q2_values:
            for L1 in L1_values:
                for L2 in L2_values:
                    if L1 + L2 >= 100:
                        continue
                    all_metrics = {'TAT': [], 'waiting_times': [],
'response_times': [], 'throughputs': []}
                    for _ in range(num_simulations):
                        processes = generate_processes(N, M)
                        stats = mlfq_scheduler(processes, Q1, Q2, L1, L2)
                        metrics = calculate_metrics(stats)
                        all_metrics['TAT'].append(metrics[0])
                        all_metrics['waiting_times'].append(metrics[1])
                        all_metrics['response_times'].append(metrics[2])
                        all_metrics['throughputs'].append(metrics[3])

                    avg_metrics = (
                        np.mean(all_metrics['TAT']),
                        np.mean(all_metrics['waiting_times']),
                        np.mean(all_metrics['response_times']),
                        np.mean(all_metrics['throughputs'])
                    )
                    results.append((Q1, Q2, L1, L2, avg_metrics))
    return results

def plot_results(results):
    fig, axs = plt.subplots(2, 2, figsize=(20, 12))
    for result in results:
        Q1, Q2, L1, L2, metrics = result
        label = f"Q1={Q1}, Q2={Q2}, L1={L1}, L2={L2}"
        axs[0, 0].scatter(Q1, metrics[0], alpha=0.6, label=label)
        axs[0, 1].scatter(Q1, metrics[1], alpha=0.6, label=label)
        axs[1, 0].scatter(Q1, metrics[2], alpha=0.6, label=label)
        axs[1, 1].scatter(Q1, metrics[3], alpha=0.6, label=label)

    axs[0, 0].set_title('Avg TAT vs Q1')
    axs[0, 1].set_title('Avg Waiting Time vs Q1')
    axs[1, 0].set_title('Avg Response Time vs Q1')
    axs[1, 1].set_title('Throughput vs Q1')
    for ax in axs.flat:
        ax.set_xlabel('Q1')
        ax.set_ylabel('Metrics')

    handles, labels = axs[0, 0].get_legend_handles_labels()
    fig.legend(handles, labels, loc='upper right', bbox_to_anchor=(1.15, 1))
    plt.tight_layout(rect=[0, 0, 0.85, 1])
    plt.show()

N = 1000
M = 100
num_simulations = 10
Q1_values = [10, 20, 30, 40, 50]
Q2_values = [20, 40, 60, 80, 100]
L1_values = [10, 20, 30, 40, 50]
L2_values = [20, 40, 60, 80]

results = run_simulations(num_simulations, N, M, Q1_values, Q2_values, L1_values,
L2_values)

plot_results(results)
```

Graphs:



B) Consider the same setup as above but now in the third level both FCFS and SJF are running. Therefore, the time assigned for the third level is split in two using a Q percentage. Therefore, we run first SJF for T% of the time assigned in the third level and for (100-T)% we run second FCFS.
Present AGAIN all your results in detail in tables and graphs The code should be in JAVA/C++ /or PYTHON (preferably in Python). Each level i of the three level feedback queue above is assigned AGAIN a time Li. Therefore, if we assume 100 ms of time the, the first level should run for L1, the second for L2 and the third one for 100-L1-L2. L1, L2 are again independent variables. In conclusion you should present graphs and tables of the following criteria

- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

With respect now to Q1, Q2, L1, L2 T.

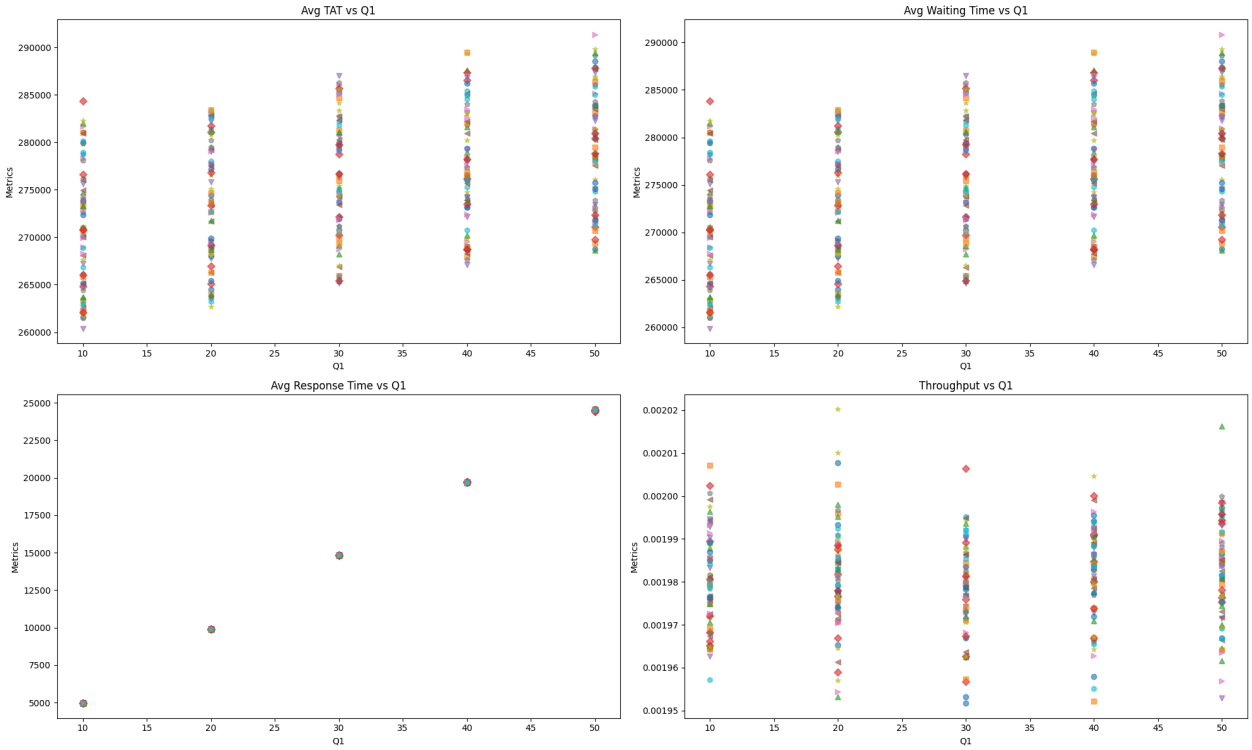
Python:

```
import random
import queue
import numpy as np
import matplotlib.pyplot as plt

class Process:
    def __init__(self, pid, arrival_time, burst_time):
        self.pid = pid
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.remaining_time = burst_time
        self.start_time = None
        self.completion_time = None

    def __repr__(self):
        return f"PID={self.pid}, AT={self.arrival_time}, BT={self.burst_time},
RT={self.remaining_time}"

def generate_processes(N, M):
    processes = []
    for pid in range(N):
        arrival_time = random.randint(0, M)
        burst_time = random.randint(10, 1000)
        processes.append(Process(pid, arrival_time, burst_time))
    return processes

def mlfq_scheduler(processes, Q1, Q2, L1, L2):
    time = 0
    completed_processes = 0
    level_queues = [queue.Queue(), queue.Queue(), queue.Queue()]
    stats = {'TAT': [], 'waiting_times': [], 'response_times': []}
    process_queue = sorted(processes, key=lambda p: p.arrival_time)

    while completed_processes < len(processes):
        while process_queue and process_queue[0].arrival_time <= time:
            level_queues[0].put(process_queue.pop(0))

        for level in range(3):
            if not level_queues[level].empty():
                current_process = level_queues[level].get()
                if current_process.start_time is None:
                    current_process.start_time = time

                time_slice = [Q1, Q2, current_process.remaining_time][level]
                execution_time = min(time_slice, current_process.remaining_time)
                current_process.remaining_time -= execution_time
                time += execution_time

                if current_process.remaining_time == 0:
                    current_process.completion_time = time
                    completed_processes += 1
                    turnaround_time = current_process.completion_time -
current_process.arrival_time
                    waiting_time = turnaround_time - current_process.burst_time
                    response_time = current_process.start_time -
current_process.arrival_time
                    stats['TAT'].append(turnaround_time)
                    stats['waiting_times'].append(waiting_time)
```

```
                    stats['response_times'].append(response_time)
                else:
                    next_level = min(level + 1, 2)
                    level_queues[next_level].put(current_process)

                break
        else:
            time += 1
    return stats


def calculate_metrics(stats):
    avg_turnaround_time = np.mean(stats['TAT'])
    avg_waiting_time = np.mean(stats['waiting_times'])
    avg_response_time = np.mean(stats['response_times'])
    throughput = len(stats['TAT']) / max(stats['TAT'])
    return avg_turnaround_time, avg_waiting_time, avg_response_time, throughput


def run_simulations(num_simulations, N, M, Q1_values, Q2_values, L1_values,
L2_values):
    results = []
    for Q1 in Q1_values:
        for Q2 in Q2_values:
            for L1 in L1_values:
                for L2 in L2_values:
                    if L1 + L2 >= 100:
                        continue
                    all_metrics = {'TAT': [], 'waiting_times': [],
'response_times': [], 'throughputs': []}
                    for _ in range(num_simulations):
                        processes = generate_processes(N, M)
                        stats = mlfq_scheduler(processes, Q1, Q2, L1, L2)
                        metrics = calculate_metrics(stats)
                        all_metrics['TAT'].append(metrics[0])
                        all_metrics['waiting_times'].append(metrics[1])
                        all_metrics['response_times'].append(metrics[2])
                        all_metrics['throughputs'].append(metrics[3])

                    avg_metrics = (
                        np.mean(all_metrics['TAT']),
                        np.mean(all_metrics['waiting_times']),
                        np.mean(all_metrics['response_times']),
                        np.mean(all_metrics['throughputs'])
                    )
                    results.append((Q1, Q2, L1, L2, avg_metrics))
    return results


def plot_results(results):
    fig, axs = plt.subplots(2, 2, figsize=(20, 12))
    markers = ['o', 's', '^', 'D', 'v', '<', '>', 'p', '*', 'h']
    for idx, result in enumerate(results):
        Q1, Q2, L1, L2, metrics = result
        label = f"Q1={Q1}, Q2={Q2}, L1={L1}, L2={L2}"
        marker = markers[idx % len(markers)]
        axs[0, 0].scatter(Q1, metrics[0], alpha=0.6, label=label, marker=marker)
        axs[0, 1].scatter(Q1, metrics[1], alpha=0.6, label=label, marker=marker)
        axs[1, 0].scatter(Q1, metrics[2], alpha=0.6, label=label, marker=marker)
        axs[1, 1].scatter(Q1, metrics[3], alpha=0.6, label=label, marker=marker)

    axs[0, 0].set_title('Avg TAT vs Q1')
```

```
    axs[0, 1].set_title('Avg Waiting Time vs Q1')
    axs[1, 0].set_title('Avg Response Time vs Q1')
    axs[1, 1].set_title('Throughput vs Q1')
    for ax in axs.flat:
        ax.set_xlabel('Q1')
        ax.set_ylabel('Metrics')

    handles, labels = axs[0, 0].get_legend_handles_labels()
    fig.legend(handles, labels, loc='upper center', bbox_to_anchor=(0.5, -0.05),
ncol=4)
    plt.tight_layout()
    plt.show()


N = 1000
M = 100
num_simulations = 10
Q1_values = [10, 20, 30, 40, 50]
Q2_values = [20, 40, 60, 80, 100]
L1_values = [10, 20, 30, 40, 50]
L2_values = [20, 40, 60, 80]


results = run_simulations(num_simulations, N, M, Q1_values, Q2_values, L1_values,
L2_values)


plot_results(results)
```

Graphs:



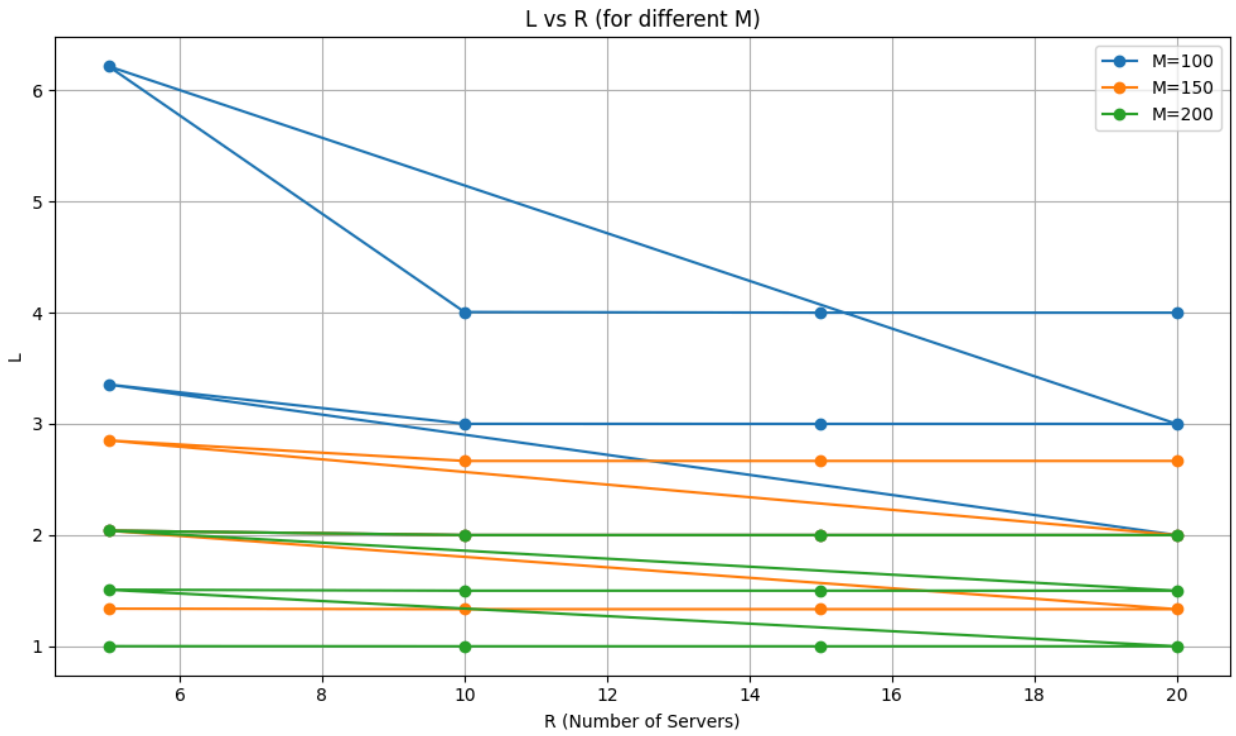
3) Consider again N processes that are produced randomly with RANDOM Burst times (in the range 10 to 1000 time units) and arriving at random moments within a range of time 0 to M time units For instance, 1000 processes arrived within 0..100 time units. N and M should vary, and they are independent variables. Suppose the same setup is repeated, after M time units have ended, with different N, M values again randomly selected. This process continues forever. Suppose finally that the system can SERVE up to R processes per time unit. The processes that exceed this number R should be postponed according to FCFS algorithm in order to be served. In this given setup calculate all quantities of M/M/1 and M/M/S queuing models. The population of processes is obviously infinite. You will calculate the averages of all known quantities when time goes to infinity. Also, calculate the same quantities when N, M, R are constant OVER TIME. In this case, draw all diagrams of how M/M/1, M/M/S quantities and probabilities vary with M, N, R. Is there any dependence on the random function used?? Change the random function used and show the differences.

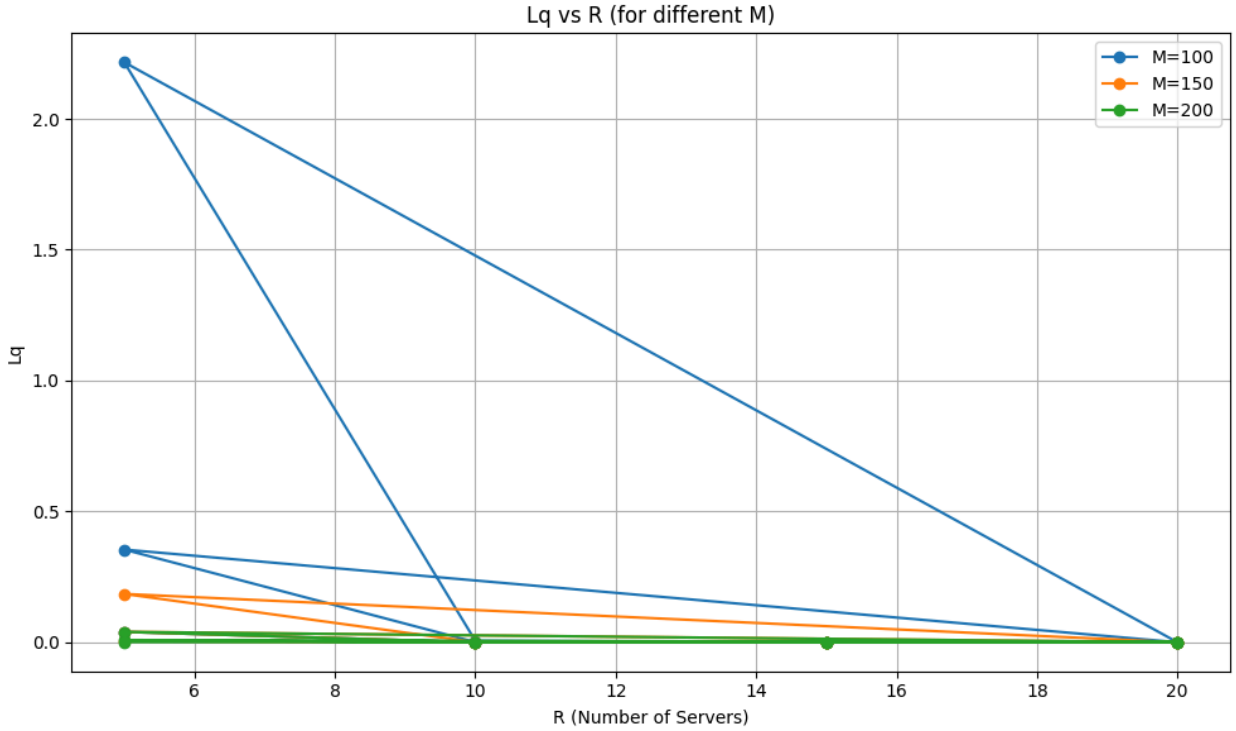
Python:

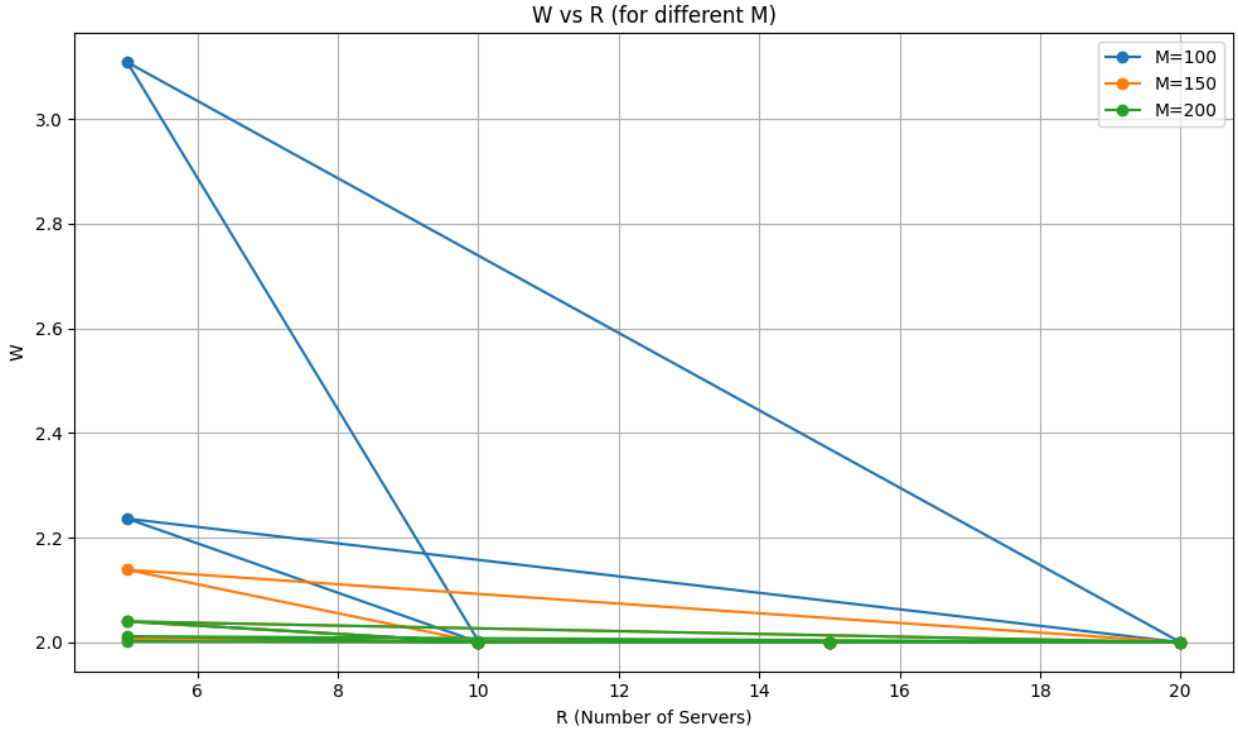
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

# M/M/S Queue metrics
def mms_metrics(lambda_rate, mu, s):
    rho = lambda_rate / (s * mu)
    if rho >= 1:
        return None

    def erlang_c(lambda_, mu_, s_):
        rho_ = lambda_ / (s_ * mu_)
        summation = sum((lambda_ / mu_)**n / math.factorial(n) for n in
range(s_))
        P0 = 1 / (summation + ((lambda_ / mu_)**s_ / (math.factorial(s_) * (1
- rho_))))
        Pq = ((lambda_ / mu_)**s_ * P0) / (math.factorial(s_) * (1 - rho_))
        return Pq

    Pq = erlang_c(lambda_rate, mu, s)
    Lq = Pq * lambda_rate / (s * mu - lambda_rate)
    Wq = Lq / lambda_rate
    W = Wq + 1 / mu
    L = lambda_rate * W
    return {'rho': rho, 'L': L, 'Lq': Lq, 'W': W, 'Wq': Wq}

# Simulation setup
N_values = [100, 150, 200]
M_values = [100, 150, 200]
R_values = [5, 10, 15, 20]
mu = 0.5  # Service rate: 0.5 processes per unit time

# Collect results
data = []
for N in N_values:
```

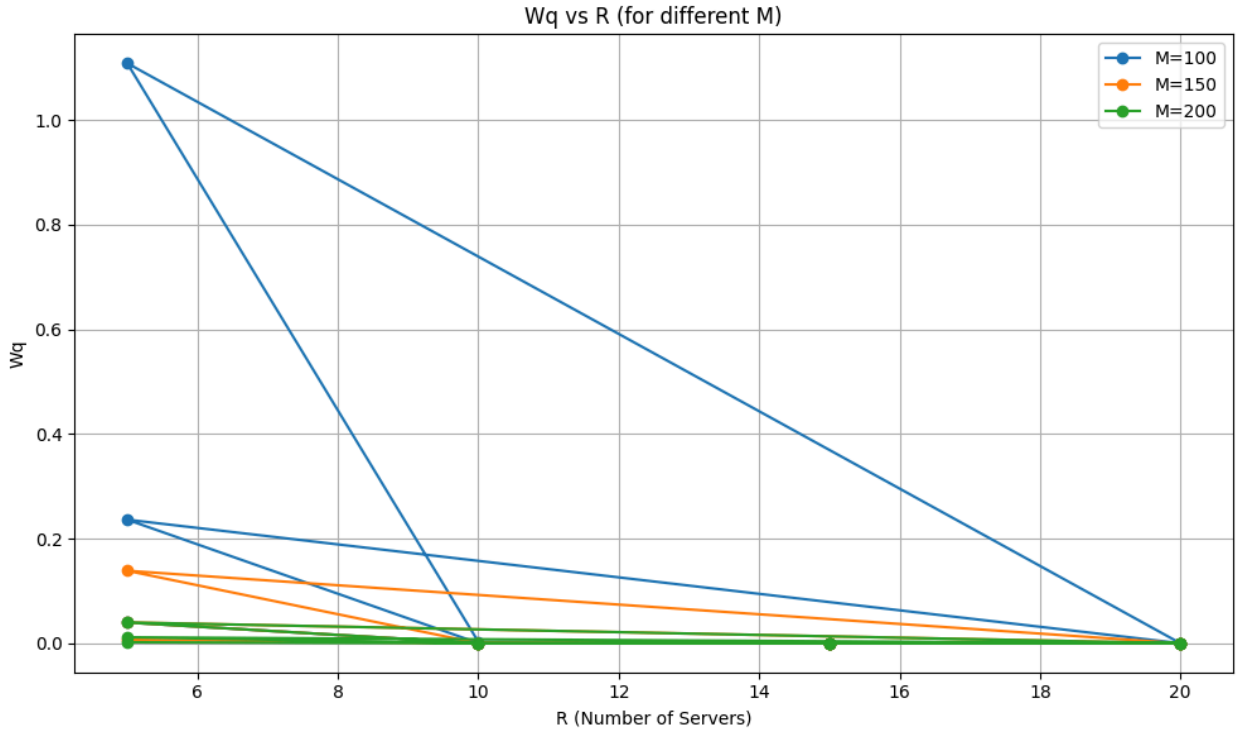
```
    for M in M_values:
        for R in R_values:
            lambda_rate = N / M
            metrics = mms_metrics(lambda_rate, mu, R)
            if metrics:
                row = {
                    'N': N,
                    'M': M,
                    'R': R,
                    'lambda': lambda_rate,
                    'rho': metrics['rho'],
                    'L': metrics['L'],
                    'Lq': metrics['Lq'],
                    'W': metrics['W'],
                    'Wq': metrics['Wq']
                }
                data.append(row)

# Create DataFrame
df = pd.DataFrame(data)

# Plotting function
def plot_metric(metric):
    fig, ax = plt.subplots(figsize=(10, 6))
    for M in sorted(df['M'].unique()):
        subset = df[df['M'] == M]
        ax.plot(subset['R'], subset[metric], marker='o', label=f'M={M}')
    ax.set_title(f'{metric} vs R (for different M)')
    ax.set_xlabel('R (Number of Servers)')
    ax.set_ylabel(metric)
    ax.legend()
    ax.grid(True)
    plt.tight_layout()
    plt.show()

# Plot selected metrics
plot_metric('L')
plot_metric('Lq')
plot_metric('W')
plot_metric('Wq')
plot_metric('rho')
```

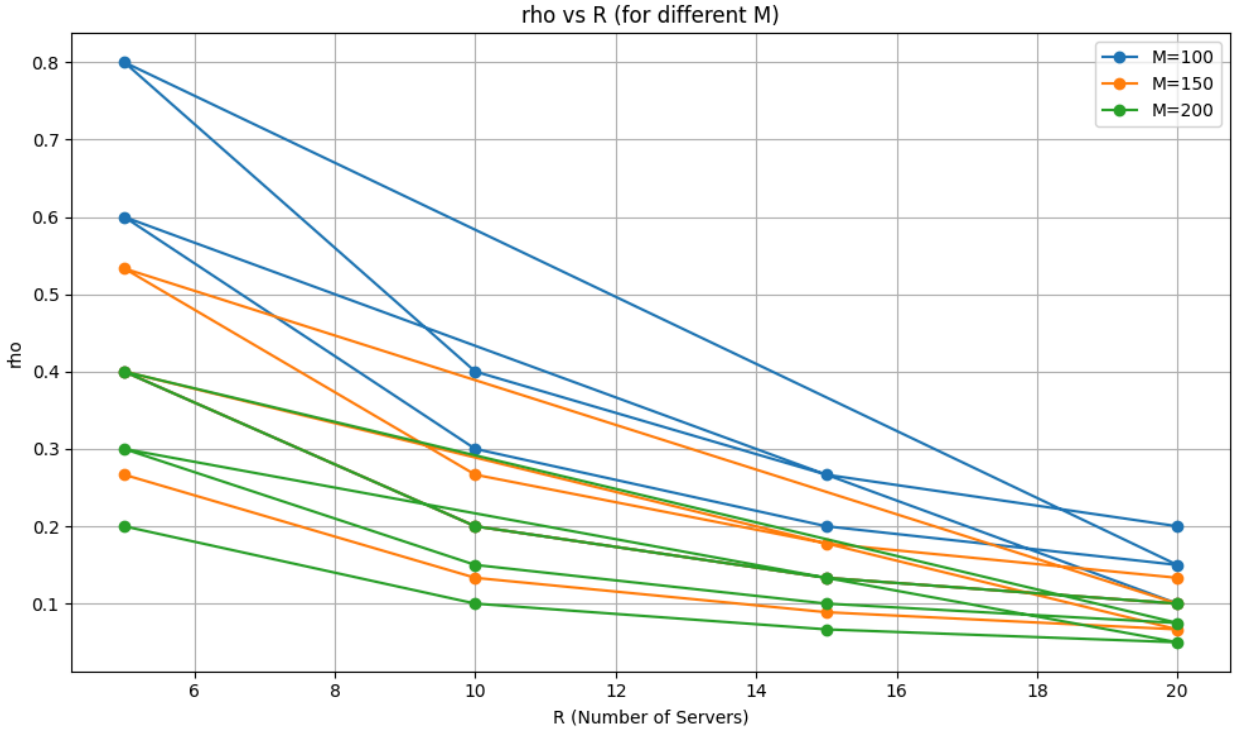
Graphs:


L vs R (for different M)
M=100
M=150
M=200
L
R (Number of Servers)
1
2
3
4
5
6
6
8
10
12
14
16
18
20


Lq vs R (for different M)
M=100
M=150
M=200
Lq
R (Number of Servers)
0.0
0.5
1.0
1.5
2.0
6
8
10
12
14
16
18
20


W vs R (for different M)
M=100
M=150
M=200
W
3.0
2.8
2.6
2.4
2.2
2.0
6
8
10
12
14
16
18
20
R (Number of Servers)


Wq vs R (for different M)
M=100
M=150
M=200
Wq
1.0
0.8
0.6
0.4
0.2
0.0
6
8
10
12
14
16
18
20
R (Number of Servers)


rho vs R (for different M)
M=100
M=150
M=200
rho
R (Number of Servers)
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
6
8
10
12
14
16
18
20