Assignment - 2

Akhil P Dominic
MT23013

1. Create a program to iteratively find the nth Fibonacci number. The value for n should be set as a
parameter (e.g., a programmer defined constant).
The formula for computing Fibonacci is as follows: fibonacci(n) = { n if n=0 or n=1
fibonacci(n−2) +
fibonacci(n−1) if n≥2

```
  GNU nano 7.2                                        fib.asm
segment .data
text db "Hello assembly",10
num1 dq 5

segment .text
global _start

_start:
mov rax,1
mov rdi,1
mov rsi,text
mov rdx,15
syscall

mov rax,0
mov rbx,1
mov r9,rax
add r9,rbx

mov r8,1_

cmp rcx,0
je zero

cmp rcx,1
je one


fibcheck:
inc r8
add rax,rbx
mov rbx,r9

mov r9,rax
add r9,rbx
cmp rax,5
jl fibcheck



mov rax,60
mov rdi,0
syscall
```

<Tried but couldnt complete>

3. Write a c program tail -n which will print last n lines of the input. The program should behave rationally no matter how much the value of n should be. Do not store the lines in 2-dimentional arrays of fixed sizes.
For correct code and execution

```
~/IIITD_SUMMER_REFRESHER_A/recursion_dir ./tail -n 3
Enter the input :
Akhil P
lorem ipsum
pen book
laptop bag
mouse keyboard


 Last 3 lines are :
pen book

laptop bag

mouse keyboard


~/IIITD_SUMMER_REFRESHER_A/recursion_dir
                                                    Ln 85, Col 38   Spaces: 4   UTF-8   LF
```

For this question, I have used the linked list approach. The program takes input from the user continuously and steps when no input is given, ie, when the user enters the “enter” key without any inputs. Whenever a line is entered, a new linked list node is created with the value as the line of text entered.On adding new lines, they are entered as linked list. Finally when we have to print the last n lines, we would calculate when to start printing by finding the value (counter-n). Then from the root we start traversing till the end of the linkedlist and and prints the last n lines.

Instructions to run:

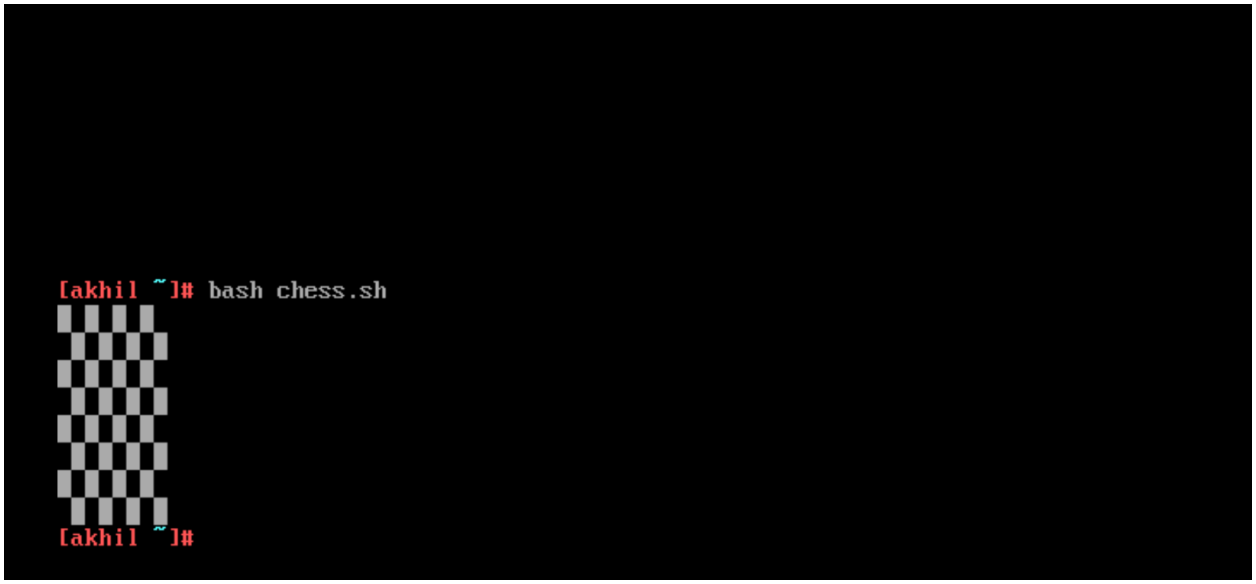
gcc tail.c -o tail
./tail -n 3

4. Write a script that will display the chessboard on the screen

```
  GNU nano 7.2                                          chess.sh

for(( i=0;i<8;i++))                             #outer for loop
do
        for((j=0;j<8;j++))                      #inner for loop
        do
                val=$(((i+j)%2))                #calculating the sum of i and j
                if [ $val -eq 0 ]               #checking if the value is even
                then
                        echo -e -n "\u2588"     #If even, printing white
                else
                        echo -e -n " "          #Else, print black
                fi
        done
        echo " "
done

                                        [ Read 15 lines ]
^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Se
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^_ Go To Line  M-E Redo       M-6 Co
```



I have used the approach where we would be having two loops running 8 times each. So we get 64 total times. The loops start from 0 to 7. Lets consider i as value of outer loop and j as the value of inner loop. Whenever the value of i+j is even, we print white and whenever the value of i+j is odd, we print black. Thus all 64 squares in the chess board can be printed.

Instructions:

Save the file in your local machine in the current directory and then run :

bash chess.sh

5. File Sorting (marks: 15)
Instructions:
Write a shell script or command-line program to perform the following tasks.
Use appropriate command-line arguments or prompts to receive inputs and display outputs.
Document your code with comments to explain the purpose and functionality of each section.
Tasks:
Prompt the user to enter the name of a directory.
Check if the directory exists. If it doesn't, display an error message and exit the program.
List all the files in the given directory.
Sort the files alphabetically.

Create a new directory named "sorted" inside the given directory.
Move each file from the original directory to the "sorted" directory.
Display a success message with the total number of files moved.
Note: Ensure proper error handling and informative error messages throughout the code.

```
  GNU nano 7.2                                          file_sorting.sh
#!/bin/bash

echo "Enter the full path of the directory : " #prompting user to enter full path
read directory_name

#reading directory name

if [ -d "$directory_name" ];
#checking if directory name exist

then
        sorted_f=$(ls $directory_name | sort)           #sorting the files in directory
        mkdir -p sorted                                 #making a directory named sorted
        count=0
        echo "The files in the directory are : "
        echo "$(ls -al $directory_name)"                #listing all files in the directory
        for i in $sorted_f;                             #Looping through all files in the directory
        do
                echo "Moving file $directory_name/$i to sorted"
                mv $directory_name/$i sorted            #moving all the files to sorted directory
                count=$(( count+1 ))                    #increment count
        done

        if [ $count -eq 0 ]
        then
                echo "No files in directory to move"

        else
                echo "Success !!!"                              #success message
        fi
        echo "Moved $count files"
else
        echo "$directory not found"                     #dir not found message
fi

                                                [ Read 34 lines ]
^G Help      ^O Write Out   ^W Where Is    ^K Cut       ^T Execute   ^C Location   M-U Undo     M-A Se
^X Exit      ^R Read File   ^\ Replace     ^U Paste     ^J Justify   ^_ Go To Line M-E Redo     M-6 Co
```

```
[akhil ~]# rm -r sorted
[akhil ~]# cd dir_file_sort/
[akhil dir_file_sort]# touch orange.txt
[akhil dir_file_sort]# touch bat.txt
[akhil dir_file_sort]# touch mango.txt
[akhil dir_file_sort]# touch xyz.txt
[akhil dir_file_sort]# touch abc.txt
[akhil dir_file_sort]# cd ..
[akhil ~]# bash file_sorting.sh
Enter the full path of the directory :
./dir_file_sort
The files in the directory are :
total 8
drwxr-xr-x 2 root root 4096 Jul 26 22:11 .
drwxr-x--- 7 root root 4096 Jul 26 22:11 ..
-rw-r--r-- 1 root root    0 Jul 26 22:11 abc.txt
-rw-r--r-- 1 root root    0 Jul 26 22:11 bat.txt
-rw-r--r-- 1 root root    0 Jul 26 22:11 mango.txt
-rw-r--r-- 1 root root    0 Jul 26 22:11 orange.txt
-rw-r--r-- 1 root root    0 Jul 26 22:11 xyz.txt
Moving file ./dir_file_sort/abc.txt to sorted
Moving file ./dir_file_sort/bat.txt to sorted
Moving file ./dir_file_sort/mango.txt to sorted
Moving file ./dir_file_sort/orange.txt to sorted
Moving file ./dir_file_sort/xyz.txt to sorted
Success !!!
Moved 5 files
[akhil ~]# ls sorted
abc.txt  bat.txt  mango.txt  orange.txt  xyz.txt
[akhil ~]# ls dir_file_sort/
[akhil ~]#
```

My approach was fairly straightforward. I prompted the user to enter the name of the directory which contains the files to be moved. If the directory name existed, then we would sort all the files in the directory and also create a directory named sorted. If the directory didn't exist, an error message was printed. Then I displayed all the files in the directory entered by the user. After that, we loop through all the files in the user-entered directory and move all the files to the directory named sorted. After that, a success message is printed.

Instructions:

bash file_sorting.sh

The directory entered by the user must exist and it should have files.

6. You are given a directory named "logs" that contains a set of log files. Each log file has a name in the format “log_YYYYMMDD.txt”, where "YYYY" represents the year, "MM" represents the month, and "DD" represents the day. The log files contain entries in the following format:
Directory: log_folder
Download this folder, unzip it, and then perform the following tasks.
Write a Linux command or script that performs the following tasks:
1. Reads all log files in the "logs" directory.
2. Extract the timestamp and message from each log entry.
3. Filter out log entries that have a timestamp older than a given date.
4. Sort the remaining log entries in descending order based on their timestamps.
5. Writes the sorted log entries to a new file named "filtered_logs.txt" in the following format:

```
  UW PICO 5.09                                                          File: log_prog.sh

log_dir="./logs"

logs=$(ls "$log_dir")

date_num=$(echo "$(date +%F)")
cur_date=$(echo "${date_num:8:10}")


for log in $logs;
do
        count=0
        log_array=()
        for word in $(cat "$log_dir/$log");
        do

                if [ $count == 1 ]
                then
                        echo ${word:8:10}>bing_text
                        consider_word=${word:8:10}

                        if [ $cur_date -gt $consider_word ]
                        then
                                echo "Filtering file.."
                                rm "$log_dir/$log"
                        fi
                fi

                if [ $count -eq 3 ]
                then
                        log_array+=$(echo -e " \n ")
                fi
                log_array+=$(echo "$word ")
                count=$((count+1))


        done
        echo "$log_array   "
        echo -e " \n "




done
        cur_date=$(date)
        #date_val=$(date -f <(echo "$cur_date") "+%Y-%m-%d %H:%M:%S")
        #echo "Current date : $cur_date"


^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Pg      ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where is     ^V Next Pg      ^U UnCut Text   ^T To Spell
```

I extracted the current date using the date command. Then I searched through all the logs in the unzipped file logs. Then i was able to extract each individual dates from the files present in that folder. Finally I removed all the files which had dates lesser than the current date <DD> day.Sorted the remaining entries in the given timestamp and wrote the entries to a new file.

```
Timestamp: 2022-01-87 10:00:00
 Message: Log entry 87

Timestamp: 2022-01-88 10:00:00
 Message: Log entry 88

Timestamp: 2022-01-89 10:00:00
 Message: Log entry 89

Timestamp: 2022-01-90 10:00:00
 Message: Log entry 90

Timestamp: 2022-01-91 10:00:00
 Message: Log entry 91

Timestamp: 2022-01-92 10:00:00
 Message: Log entry 92

Timestamp: 2022-01-93 10:00:00
 Message: Log entry 93

Timestamp: 2022-01-94 10:00:00
 Message: Log entry 94

Timestamp: 2022-01-95 10:00:00
 Message: Log entry 95

Timestamp: 2022-01-96 10:00:00
 Message: Log entry 96

Timestamp: 2022-01-97 10:00:00
 Message: Log entry 97

Timestamp: 2022-01-98 10:00:00
 Message: Log entry 98

Timestamp: 2022-01-99 10:00:00
 Message: Log entry 99
```

<Completed till here>