Lab Assignment-2

Name: Abhishek Dutt Roll: MT23008

Subject: System Programming (Refresher Module)

---

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?

Ans:

```
section .data
   result_message db "The 5th Fibonacci number is: ", 0
   newline db 5
   space db " "

section .bss
   fibonacci_number resb 11

section .text
   global _start

_start:
   mov eax, 0
   mov ebx, 1
   mov ecx, 10

fibonacci_loop:
   add eax, ebx
   xchg eax, ebx
   loop fibonacci_loop


   mov ecx, fibonacci_number + 10
   mov byte [ecx], 0
   mov ebx, 10

convert_loop:
```

```
    dec ecx
    xor edx, edx
    div ebx
    add dl, '0'
    mov [ecx], dl
    test eax, eax
    jnz convert_loop
    lea edx, [fibonacci_number + 10]
    sub edx, ecx

    mov eax, 4
    mov ebx, 1
    mov ecx, result_message
    add edx, 30
    int 0x80

    mov eax, 4
    mov ebx, 1
    mov ecx, fibonacci_number
    mov edx, edx
    int 0x80

    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 1
    int 0x80


    mov eax, 1
    xor ebx, ebx
    int 0x80
```

Output:

```
55
(base) ➜  A1 nasm -f elf64 -o Q1.o Q1.asm
(base) ➜  A1 ld -o Q1 Q1.o
(base) ➜  A1 ./Q1
The 5th Fibonacci number is:  55%
(base) ➜  A1
```

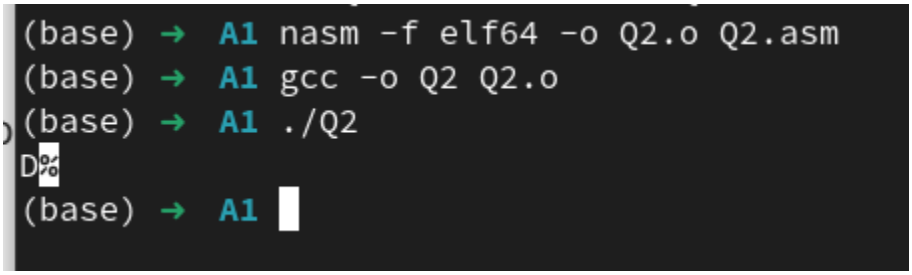
2. Simple example program to convert an integer into an ASCII string

Ans:

```
section .data
   num dd 20
   nl db 10
section .bss
   ascii resb 11
section .text
   global main
main:
   mov eax, dword [num]
   add eax, '0'
   mov byte [ascii], al
   mov byte [ascii + 1], 0
   call print
print:
   mov rsi, ascii
   mov rax, 0x1
   mov rdi, 0x1
   mov rdx, 0x1

   syscall

   mov rsi, nl
   mov rax, 0x1
   mov rdi, 0x1
   mov rdx, 0x1

   call end
end:
   mov eax, 60
   xor edi, edi
```

```
    syscall
```

Output:



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.

Ans:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

#define MAX_LINE_LENGTH 1024

void printLastLines(int fd, const char *filename, int numLines) {
    int i, n, m, totalLines = 0, startLine;

    char buffer[MAX_LINE_LENGTH];

    while ((n = read(fd, buffer, sizeof(buffer))) > 0) {
        for (i = 0; i < n; i++) {
            if (buffer[i] == '\n') {
                totalLines++;
                if (strcmp(filename, "") == 0) {
                    printf("\n");
                }
            } else {
                if (strcmp(filename, "") == 0) {
```

```
                if (buffer[i] == '\0') {
                    return;
                }
                if (buffer[i] != '\n') {
                    printf("%c", buffer[i]);
                } else {
                    printf("\n");
                }
            }
        }
    }
}

close(fd);

startLine = totalLines - numLines;
int lineCount = 0;
int fd2 = open(filename, 0);

while ((m = read(fd2, buffer, sizeof(buffer))) > 0) {
    for (i = 0; i < m; i++) {
        if (buffer[i] == '\n') {
            lineCount++;
        }
        if (lineCount >= startLine) {
            if (buffer[i] != '\n' && lineCount >= startLine) {
                printf("%c", buffer[i]);
            } else {
                printf("\n");
                lineCount++;
            }
        }
    }
}

close(fd2);

if (n < 0) {
    printf("tail: error while reading\n");
    return;
```

```
    }
}

int main(int argc, char *argv[]) {
    int fd, i;

    if (argc <= 1) {
        printLastLines(0, "", 10);
        return 1;
    } else if (argc == 2) {
        for (i = 1; i < argc; i++) {
            if ((fd = open(argv[i], 0)) < 0) {
                printf("Error opening the file: %s\n", argv[i]);
                return 1;
            }
            printLastLines(fd, argv[i], 10);
            close(fd);
        }
    } else if (argc == 3) {
        char numStr[MAX_LINE_LENGTH];
        strcpy(numStr, argv[1]);
        char *numArg = numStr + 1;
        int numLines = atoi(numArg);

        for (i = 2; i < argc; i++) {
            if ((fd = open(argv[i], 0)) < 0) {
                printf("Error opening the file: %s\n", argv[i]);
                return 1;
            }
            printLastLines(fd, argv[i], numLines);
            close(fd);
        }
    } else {
        printf("Usage: %s [-x] <filename>\n", argv[0]);
    }

    return 0;
}
```

Output:

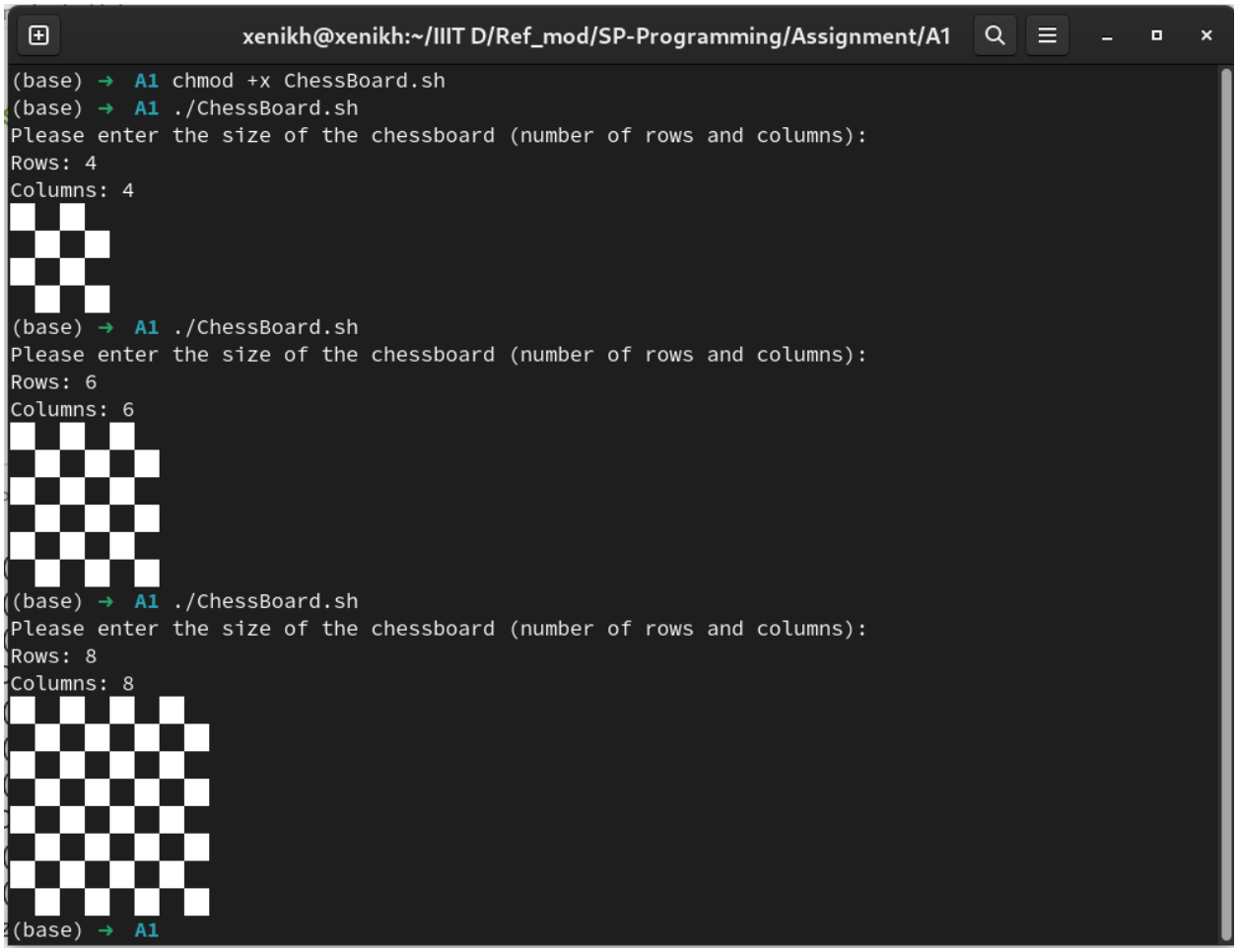
```
(base) ➜  A1 gcc tail_n.c -o tail-n
(base) ➜  A1 ./tail-n -2 test.txt

asdgjasdhfkljasdhflasdh
asdf
(base) ➜  A1 ./tail-n -4 test.txt

asdhflasdf
asdfklashdfl
asdgjasdhfkljasdhflasdh
asdf
(base) ➜  A1 ./tail-n -15 test.txt
whfklasdhklf
asdfasdf
asdfasdf
asfh
gasdfg
asdf
asdfg
asdhflasdf
asdfklashdfl
asdgjasdhfkljasdhflasdh
asdf
```

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

```
#!/bin/bash

print_chessboard() {
    local rows=$1
    local columns=$2
    local black_square="  "
    local white_square="  "

    for ((i = 1; i <= rows; i++)); do
        for ((j = 1; j <= columns; j++)); do
            if (( (i + j) % 2 == 0 )); then
                printf "$black_square"
```

```
            else
                printf "$white_square"
            fi
        done
        printf "\n"
    done
}

validateInput() {
    if [[ "$1" =~ ^[1-9][0-9]*$ ]]; then
        return 0
    else
        return 1
    fi
}

echo "Please enter the size of the chessboard (number of rows and
columns):"
read -p "Rows: " input_rows
read -p "Columns: " input_columns

if ! validateInput "$input_rows" || ! validateInput "$input_columns"; then
    echo "Error: Invalid input. Please enter positive integers for the
number of rows and columns."
    exit 1
fi

print_chessboard "$input_rows" "$input_columns"
```

Output:



5. File Sorting
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.

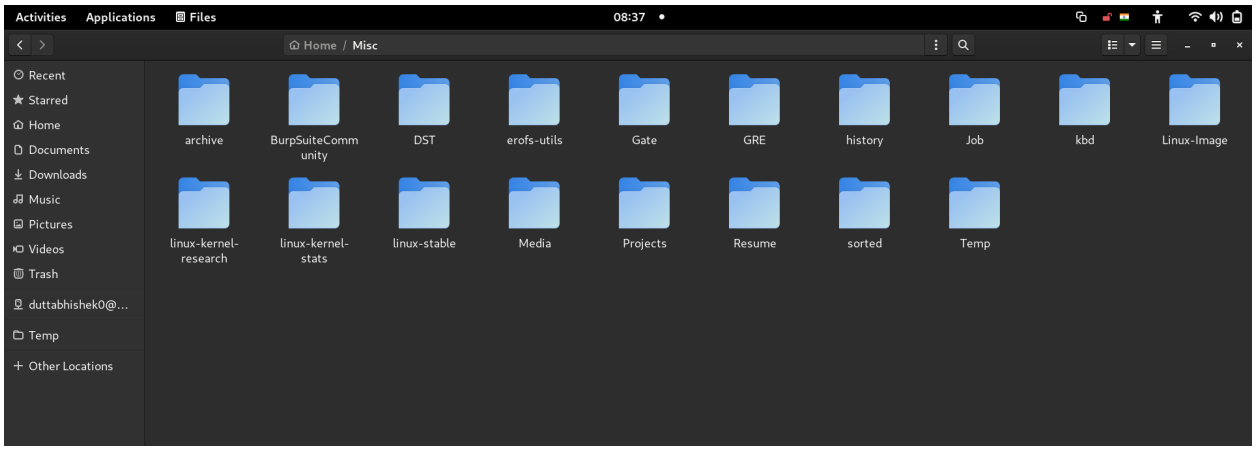
Ans:

```
#!/bin/bash

echo "Enter the directory name: "
read directName
```

```
# Check if the directory is valid or not
if [ ! -d "$directName" ];then
   echo "Directory doesn't exist"
   exit 1
fi

# List the directories
echo "Directories under $directName"
ls "$directName"

# Using the -v flag to sort the files alphabetically
echo -e "Files in sorted order: "
ls -v "$directName"

# Name the new directory
newDirect="$directName/sorted"

# Make the directory
mkdir -p "$newDirect"

cnt=0
# Count the number of files in such a directory.
for file in "$directName"/*; do
   if [ -f "$file" ] && [ "$file" != "$newDirect" ]; then
       mv "$file" "$newDirect"
       ((cnt++))
   fi
done

echo -e "\n Success: $cnt files moved "
```

Output:

```
(base) ➜  A1 ./Q5.sh
Enter the directory name :
/home/xenikh/Misc
Directories under /home/xenikh/Misc
archive            erofs-utils  history  Linux-Image           linux-stable  releaseTime.csv  Temp
BurpSuiteCommunity  Gate         Job      linux-kernel-research  Media         res.sh           test.py
DST                GRE          kbd      linux-kernel-stats     Projects      Resume           test.sh
Files in sorted order:
BurpSuiteCommunity  Gate         Media     Temp         history               linux-kernel-stats  res.sh
DST                 Job          Projects  archive      kbd                   linux-stable        test.py
GRE                 Linux-Image  Resume    erofs-utils  linux-kernel-research  releaseTime.csv     test.sh

 Success: 4 files moved
(base) ➜  A1
```



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:

Ans:

```bash
#!/bin/bash
# Handle error message
if [ $# -ne 1 ]; then
   echo "Usage: $0 <date>"
   exit 1
fi

# Store the date to be filtered inside a variable
filter_date=$(date -d "$1" "+%s")
output_file="filtered_logs.txt"

# Check for the directory
if [ ! -d "logs" ]; then
   echo "Directory 'logs' not found."
```

```
    exit 1
fi

echo "Filtering logs older than $1..."

# Method to filter out the logs depending on the date.
awk -F ': ' -v filter_date="$filter_date" '
    function get_timestamp(str) {
        split(str, date_time, /[-: ]/);
        return mktime(date_time[1]" "date_time[2]" "date_time[3]"
"date_time[4]" "date_time[5]" "date_time[6]);
    }
    BEGIN { OFS = FS }
    /^Timestamp:/ {
        timestamp = get_timestamp($2);
        in_range = (timestamp >= filter_date);
    }
    in_range { print }
' logs/log_* | sort -r -k2 > "$output_file"

echo "Filtered logs written to $output_file."
```

Output:

```
(base) ➜  A1 ./Q6.sh 20220102
Filtering logs older than 20220102...
Filtered logs written to filtered_logs.txt.
(base) ➜  A1
```