	Registration Number	Surname	Forename	% Contribution
Student 1	001090189	Chiurcci	Cristina	25%
Student 2	001131628	Chavush	Chisel	25%
Student 3	001167783	Dumbravanu	Cristian	25%
Student 4	001166012	Chowdhury	Mutammim	25%

TASK		%	Mark
TASK 1	Introduction the Command Line Interface (CLI)	20%	
TASK 2	Running a C program in the CLI	15%	
TASK 3	Running an assembly program in the CLI	15%	
TASK 4	Shell Script Programming	50%	
Total		100%	

TASK 1

The very first thing we did was using our first command **cd**, which stands for Change Directory. We did this to set the pathway, where all the operations will occur, and from which we will take all of the files we are going to work with.

```
cc4201c@stulinux-02:~
File  Edit  View  Search  Terminal  Help
[cc4201c@stulinux-02 ~]$
```

cd Documents

To check whether there are an files or directories in our current location, we used the next command **ls**, which stands for list. Upon calling this command, we saw the unixlaboratory directory. So our next step was changing the directory again.

ls

cd unixlab.

```
cc4201c@stulinux-02:~/Documents/unixlab.
File  Edit  View  Search  Terminal  Help
[cc4201c@stulinux-02 ~]$ cd Documents
[cc4201c@stulinux-02 Documents]$ ls
unixlab.  unixlaboratuary
[cc4201c@stulinux-02 Documents]$ cd unixlab.
[cc4201c@stulinux-02 unixlab.]$ ls
GregorianCalendar.txt  june3k.txt    sh3.sh     Task4a.sh   task4d.sh
hello_asm              menu.sh       sh4.sh     Task4a.txt  task4e.sh
hello.asm              owen.txt      task1.sh   task4b      task4e.sh~
hello.asm.o            poemtemp.txt  task1.txt  task4b.sh   Task4.txt
hello.c                poem.txt      task2.txt  task4c.sh   try
hello.c~               sh1.sh        task3      task4c.sh~  typescript
hello.out              sh1.sh~       task3.txt  task4d      users.sh
june3000.txt           sh2.sh        Task4a     task4d~
[cc4201c@stulinux-02 unixlab.]$
```

We used the **ls** command again, and we found all of the files we downloaded earlier.
We then proceeded with the **script** command. Script is a computer program designed to be run by the Unix Shell, a command line interpreter. So our script's life cycle began with the following command:

script task1

```
[cc4201c@stulinux-02 unixlab.]$ script task1
Script started, file is task1
[cc4201c@stulinux-02 unixlab.]$
```

As per instructions provided, we examined the following commands:

cal 3000 – after executing this command, the full calendar of the year 3000 appeared in the terminal.

```
cc4201c@stulinux-02:~/Documents/unixlab.
File  Edit  View  Search  Terminal  Help
                              3000

       January               February                March
Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
       1  2  3  4  5                   1  2                   1  2
 6  7  8  9 10 11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9
13 14 15 16 17 18 19   10 11 12 13 14 15 16   10 11 12 13 14 15 16
20 21 22 23 24 25 26   17 18 19 20 21 22 23   17 18 19 20 21 22 23
27 28 29 30 31         24 25 26 27 28         24 25 26 27 28 29 30
                                              31
        April                  May                   June
Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6              1  2  3  4                      1
 7  8  9 10 11 12 13    5  6  7  8  9 10 11    2  3  4  5  6  7  8
14 15 16 17 18 19 20   12 13 14 15 16 17 18    9 10 11 12 13 14 15
21 22 23 24 25 26 27   19 20 21 22 23 24 25   16 17 18 19 20 21 22
28 29 30               26 27 28 29 30 31      23 24 25 26 27 28 29
                                              30
        July                  August                September
Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6                 1  2  3    1  2  3  4  5  6  7
 7  8  9 10 11 12 13    4  5  6  7  8  9 10    8  9 10 11 12 13 14
14 15 16 17 18 19 20   11 12 13 14 15 16 17   15 16 17 18 19 20 21
21 22 23 24 25 26 27   18 19 20 21 22 23 24   22 23 24 25 26 27 28
28 29 30 31            25 26 27 28 29 30 31   29 30

       October               November               December
Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
       1  2  3  4  5                   1  2    1  2  3  4  5  6  7
 6  7  8  9 10 11 12    3  4  5  6  7  8  9    8  9 10 11 12 13 14
13 14 15 16 17 18 19   10 11 12 13 14 15 16   15 16 17 18 19 20 21
20 21 22 23 24 25 26   17 18 19 20 21 22 23   22 23 24 25 26 27 28
27 28 29 30 31         24 25 26 27 28 29 30   29 30 31

     Mo Tu We Th Fr Sa Su
```

cal 6 3000 – after executing this command, the calendar for the 6th month (June), of year 3000 has appeared in the terminal.

```
[cc4201c@stulinux-02 unixlab.]$ cal 6 3000
      June 3000
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
[cc4201c@stulinux-02 unixlab.]$
```

We then proceeded to the next command, and we noticed something what seemed to be an error.

cal 9 1752 – after executing this command, the calendar for the 9th month (September) of year 1752 has appeared, however, days 3 to 13 of the month were missing. Upon further examination, we understood that this is not an error with the computer, the code or the command.
This "error" was due to the calendar switch in 1752 (Julian to Gregorian). These two calendars have different levels of accuracy, compared to the solar year, as well as different numbers of days in months, and even different numbers of months in some years (intercalary years in Julian calendar).The total number of days per year is 355 or 377-378 (Julian) and 365-366 (Gregorian), hence the "error".

```
[cc4201c@stulinux-02 unixlab.]$ cal 9 1752
   September 1752
Mo Tu We Th Fr Sa Su
    1  2 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30


[cc4201c@stulinux-02 unixlab.]$
```

Next, we repeated the cal 6 3000 command, but this time, we added the >june3000.txt redirection. As expected, nothing appeared on screen, however, the redirection has created a new file in the directory we were in. We checked this using the **ls** command again.

cal 6 3000 > june3000.txt
ls – we now saw a new file called june3000.txt

```
[cc4201c@stulinux-02 unixlab.]$ ls
GregorianCalendar.txt  hello.out     poem.txt  task1      Task4a      task4c.sh~  Task4.txt
hello_asm              june3000.txt  sh1.sh    task1.sh   Task4a.sh   task4d      try
hello.asm              june3k.txt    sh1.sh~   task1.txt  Task4a.txt  task4d~     typescript
hello.asm.o            menu.sh       sh2.sh    task2.txt  task4b      task4d.sh   users.sh
hello.c                owen.txt      sh3.sh    task3      task4b.sh   task4e.sh
hello.c~               poemtemp.txt  sh4.sh    task3.txt  task4c.sh   task4e.sh~
[cc4201c@stulinux-02 unixlab.]$ cal 6 3000 > jun3000.txt
[cc4201c@stulinux-02 unixlab.]$ ls
GregorianCalendar.txt  jun3000.txt   sh1.sh    task1.txt   task4b      task4e.sh
hello_asm              june3000.txt  sh1.sh~   task2.txt   task4b.sh   task4e.sh~
hello.asm              june3k.txt    sh2.sh    task3       task4c.sh   Task4.txt
hello.asm.o            menu.sh       sh3.sh    task3.txt   task4c.sh~  try
hello.c                owen.txt      sh4.sh    Task4a      task4d      typescript
hello.c~               poemtemp.txt  task1     Task4a.sh   task4d~     users.sh
hello.out              poem.txt      task1.sh  Task4a.txt  task4d.sh
[cc4201c@stulinux-02 unixlab.]$
```

We proceeded to the next instruction:

cp june3000.txt june3k.txt

```
[cc4201c@stulinux-02 unixlab.]$ cp june3000.txt june3k.txt
[cc4201c@stulinux-02 unixlab.]$ ls
GregorianCalendar.txt  jun3000.txt   sh1.sh    task1.txt   task4b      task4e.sh
hello_asm              june3000.txt  sh1.sh~   task2.txt   task4b.sh   task4e.sh~
hello.asm              june3k.txt    sh2.sh    task3       task4c.sh   Task4.txt
hello.asm.o            menu.sh       sh3.sh    task3.txt   task4c.sh~  try
hello.c                owen.txt      sh4.sh    Task4a      task4d      typescript
hello.c~               poemtemp.txt  task1     Task4a.sh   task4d~     users.sh
hello.out              poem.txt      task1.sh  Task4a.txt  task4d.sh
[cc4201c@stulinux-02 unixlab.]$
```

The **cp** command stands for copy.
Upon calling the **ls** command again, we noticed both june3000.txt and june3k.txt files. The **cp** command created a copy of june3000.txt in a new file called june3k.txt.
We called the **ls -l** command. The permission flag **-l** gave us additional information about the listed files, such as: size, date and time the files were created.
Next, we displayed the file owen.txt using **cat** command, derived from concatenate.

cc4201c@stulinux-02:~/Documents/unixlab.

File Edit View Search Terminal Help

```
[cc4201c@stulinux-02 unixlab.]$ ls -l
total 380
-rw-r-----. 1 cc4201c domain users  2769 Aug 31  2020 GregorianCalendar.txt
-rwxr-xr-x. 1 cc4201c domain users   888 Sep 30 16:44 hello_asm
-rw-r-----. 1 cc4201c domain users   366 Sep 30 16:44 hello.asm
-rw-r--r--. 1 cc4201c domain users   896 Sep 30 16:44 hello.asm.o
-rw-r-----. 1 cc4201c domain users    71 Oct  5 11:16 hello.c
-rw-r-----. 1 cc4201c domain users    65 Sep 30 16:07 hello.c~
-rwxr-xr-x. 1 cc4201c domain users  8360 Sep 30 16:08 hello.out
-rw-r--r--. 1 cc4201c domain users   150 Oct  6 13:22 jun3000.txt
-rw-r--r--. 1 cc4201c domain users   150 Sep 30 15:32 june3000.txt
-rw-r--r--. 1 cc4201c domain users   150 Oct  6 13:25 june3k.txt
-rw-r-----. 1 cc4201c domain users   752 Aug 31  2020 menu.sh
-rw-r-----. 1 cc4201c domain users  1302 Aug 31  2020 owen.txt
-rw-r--r--. 1 cc4201c domain users  1109 Oct  5 10:33 poemtemp.txt
-rw-r-----. 1 cc4201c domain users  1109 Oct  5 10:34 poem.txt
-rwx------. 1 cc4201c domain users   136 Oct  4 17:12 sh1.sh
-rwx------. 1 cc4201c domain users   141 Oct  4 11:37 sh1.sh~
-rw-r-----. 1 cc4201c domain users    61 Oct 24  2016 sh2.sh
-rw-r-----. 1 cc4201c domain users    54 Aug 31  2020 sh3.sh
-rw-r-----. 1 cc4201c domain users    93 Aug 31  2020 sh4.sh
-rw-r--r--. 1 cc4201c domain users     0 Oct  6 13:09 task1
-rw-r--r--. 1 cc4201c domain users 96858 Oct  5 11:15 task1.sh
-rw-r--r--. 1 cc4201c domain users 96858 Sep 30 15:46 task1.txt
-rw-r--r--. 1 cc4201c domain users  1456 Sep 30 16:09 task2.txt
-rw-r--r--. 1 cc4201c domain users 12535 Sep 30 16:49 task3
-rw-r--r--. 1 cc4201c domain users 12535 Oct  2 14:57 task3.txt
-rw-r--r--. 1 cc4201c domain users   137 Oct  4 18:15 Task4a
-rwx------. 1 cc4201c domain users   137 Oct  5 10:52 Task4a.sh
-rwx------. 1 cc4201c domain users   137 Oct  4 17:13 Task4a.txt
-rwx------. 1 cc4201c domain users    94 Oct  4 17:54 task4b
-rwx------. 1 cc4201c domain users   156 Oct  4 18:10 task4b.sh
-rwx------. 1 cc4201c domain users   275 Oct  5 10:45 task4c.sh
-rwx------. 1 cc4201c domain users   243 Oct  4 19:51 task4c.sh~
```

cat owen.txt
more GregorianCalendar.txt
cat poem.txt > poem2.txt
ls -l
cat poem.txt >> poem2.txt
ls -l
more

```
cd4544i@stulinux-06:~/Documents/unixlaboratory
File  Edit  View  Search  Terminal  Help
-rwx------. 1 cd4544i domain users   188 Oct  4 18:45 Task4b~
-rw-r-----. 1 cd4544i domain users    87 Oct  1 21:49 users.sh
-rw-r-----. 1 cd4544i domain users    64 Aug 31  2020 users.sh~
[cd4544i@stulinux-06 unixlaboratory]$ cat poem.txt >> poem2.txt
[cd4544i@stulinux-06 unixlaboratory]$ ls -l
total 144
-rw-r--r--. 1 cd4544i domain users  4658 Aug 31 08:59 Files for UNIX Laboratory-20210831.zip
-rw-r-----. 1 cd4544i domain users  2769 Aug 31  2020 GregorianCalendar.txt
-rwxr-xr-x. 1 cd4544i domain users   896 Oct  1 21:17 hello_asm
-rw-r-----. 1 cd4544i domain users   374 Oct  1 21:13 hello.asm
-rw-r--r--. 1 cd4544i domain users   896 Oct  1 21:16 hello.asm.o
-rw-r-----. 1 cd4544i domain users    69 Oct  1 21:01 hello.c
-rw-r-----. 1 cd4544i domain users    68 Aug 31  2020 hello.c~
-rwxr-xr-x. 1 cd4544i domain users  8360 Oct  1 20:55 hello.out
-rw-r--r--. 1 cd4544i domain users   150 Oct  1 20:25 june3000.txt
-rw-r--r--. 1 cd4544i domain users   150 Oct  1 20:27 june3k.txt
-rw-r-----. 1 cd4544i domain users   752 Aug 31  2020 menu.sh
-rw-r-----. 1 cd4544i domain users  1302 Aug 31  2020 owen.txt
-rw-r--r--. 1 cd4544i domain users  2244 Oct  5 00:03 poem2.txt
-rw-r-----. 1 cd4544i domain users  1122 Sep  1  2020 poem.txt
-rwx------. 1 cd4544i domain users    97 Oct  2 22:14 sh1.sh
-rwx------. 1 cd4544i domain users    94 Oct  1 22:37 sh1.sh~
-rwx------. 1 cd4544i domain users    61 Oct  4 17:46 sh2.sh
-rwx------. 1 cd4544i domain users    60 Oct  2 22:43 sh2.sh~
-rwx------. 1 cd4544i domain users    54 Aug 31  2020 sh3.sh
-rwx------. 1 cd4544i domain users    93 Aug 31  2020 sh4.sh
-rw-r--r--. 1 cd4544i domain users 24300 Oct  1 20:47 task1
-rw-r--r--. 1 cd4544i domain users  2528 Oct  1 21:02 task2
-rw-r--r--. 1 cd4544i domain users  3594 Oct  1 21:18 task3
-rwx------. 1 cd4544i domain users   160 Oct  4 18:22 Task4a
-rwx------. 1 cd4544i domain users   190 Oct  4 18:50 Task4b
-rwx------. 1 cd4544i domain users   188 Oct  4 18:45 Task4b~
-rw-r-----. 1 cd4544i domain users    87 Oct  1 21:49 users.sh
-rw-r-----. 1 cd4544i domain users    64 Aug 31  2020 users.sh~
[cd4544i@stulinux-06 unixlaboratory]$
```

The **rm** command stands for remove.

rm poem2.txt
ls

After calling **ls**, we saw that poem2.txt is not in our directory anymore.

exit

And with **exit** command, the life cycle of our script ends.

TASK 2

In this task, using a sequence of commands, we have learnt how to compile, execute and edit a program in C, using the Unix terminal.
We started this script's life cycle using **script** command.

script task2

Next, we compiled the hello.c file.

gcc hello.c -o hello.out

This operation has compiled the C file, and generated a hello.out file, that we executed later on.

./hello.out

```
[cd4544i@stulinux-06 ~]$ cd Documents
[cd4544i@stulinux-06 Documents]$ cd unixlaboratory
[cd4544i@stulinux-06 unixlaboratory]$ script task2
Script started, file is task2
[cd4544i@stulinux-06 unixlaboratory]$ gcc hello.c -o hello.out
[cd4544i@stulinux-06 unixlaboratory]$ ./hello.out
Hello Cristian[cd4544i@stulinux-06 unixlaboratory]$
```

Hello Cristian appeared on screen. (Our group member's name)

In the GUI we opened the hello.c file and an editor appeared on screen.
We changed "World" to "Group 21".
We had a hypothesis, that if we execute the program without recompiling the file, nothing will change. So we proceeded with executing the program again.

./hello.out

As suspected, nothing changed, and **Hello Cristian** appeared on screen again.
We then recompiled the file and executed it again

gcc hello.c -o hello.out
./hello.out

As expected, this time we received the message:

```
File  Edit  View  Search  Terminal  Help
[cd4544i@stulinux-06 ~]$ cd Documents
[cd4544i@stulinux-06 Documents]$ cd unixlaboratory
[cd4544i@stulinux-06 unixlaboratory]$ script task2
Script started, file is task2
[cd4544i@stulinux-06 unixlaboratory]$ gcc hello.c -o hello.out
[cd4544i@stulinux-06 unixlaboratory]$ ./hello.out
Hello Cristian[cd4544i@stulinux-06 unixlaboratory]$ exit
exit
Script done, file is task2
[cd4544i@stulinux-06 unixlaboratory]$
```

To stop the recordin gof the script file has been stoped by the < exit > command.

Hello Cristian

We ended the script's life cycle using **exit** command.

TASK 3

In this task, we have worked with an assembly program.

First, we have translated the file to an executable and linkable format (ELF).

nasm -f elf64 hello.asm -o hello.asm.o

Then, the ELF formatted file was linked to the current operating system's functions.

ld hello.asm.o -o hello_asm

As expected, these commands did not display anything on screen, but as the assembly file is now translated and linked, we can now execute the program.

./hello_asm

After executing hello_asm file, "Hello World" appeared on screen.

We then proceeded to edit the **hello.asm** assembly program. We have changed the text to "Hello Group 59!".

Open

hello.asm
~/Documents/unixlab.

```
; Define variables in the data section
SECTION .DATA
        hello:     db 'Group59',10
        helloLen:  equ $-hello

; Code goes in the text section
SECTION .TEXT
        GLOBAL _start

_start:
        mov eax,4
        mov ebx,1
        mov ecx,hello
        mov edx,helloLen
        int 80h

        mov eax,1
        mov ebx,0
        int 80h
```

We have then translated and linked it again.

nasm -f elf64 hello.asm -o hello.asm.o

ld hello.asm.o -o hello_asm

./hello_asm

The executed program has written “Hello Group59! on screen”

```
cd4544i@stulinux-02:~/Documents/unixlaboratory
File  Edit  View  Search  Terminal  Help
[cd4544i@stulinux-02 ~]$ cd Documents
[cd4544i@stulinux-02 Documents]$ cd unixlaboratory
[cd4544i@stulinux-02 unixlaboratory]$ nasm -f elf64 hello.asm -o hello.asm.o
[cd4544i@stulinux-02 unixlaboratory]$ ld hello.asm.o -o hello_asm
[cd4544i@stulinux-02 unixlaboratory]$ ./hello_asm
Hello Group59!
[cd4544i@stulinux-02 unixlaboratory]$
```

We then used the **ls -l** command to display all the files in the directory, as well as information about them, such as size.

```
[cd4544i@stulinux-02 unixlaboratory]$ ls -l
total 176
-rw-r--r--. 1 cd4544i domain users  4658 Aug 31 08:59 Files for UNIX Laboratory-20210831.zip
-rw-r-----. 1 cd4544i domain users  2769 Aug 31  2020 GregorianCalendar.txt
-rwxr-xr-x. 1 cd4544i domain users   896 Oct  6 13:55 hello_asm
-rw-r-----. 1 cd4544i domain users   373 Oct  6 13:53 hello.asm
-rw-r--r--. 1 cd4544i domain users   896 Oct  6 13:54 hello.asm.o
-rw-r-----. 1 cd4544i domain users    69 Oct  1 21:01 hello.c
-rw-r-----. 1 cd4544i domain users    68 Aug 31  2020 hello.c~
-rwxr-xr-x. 1 cd4544i domain users  8360 Oct  5 00:13 hello.out
```

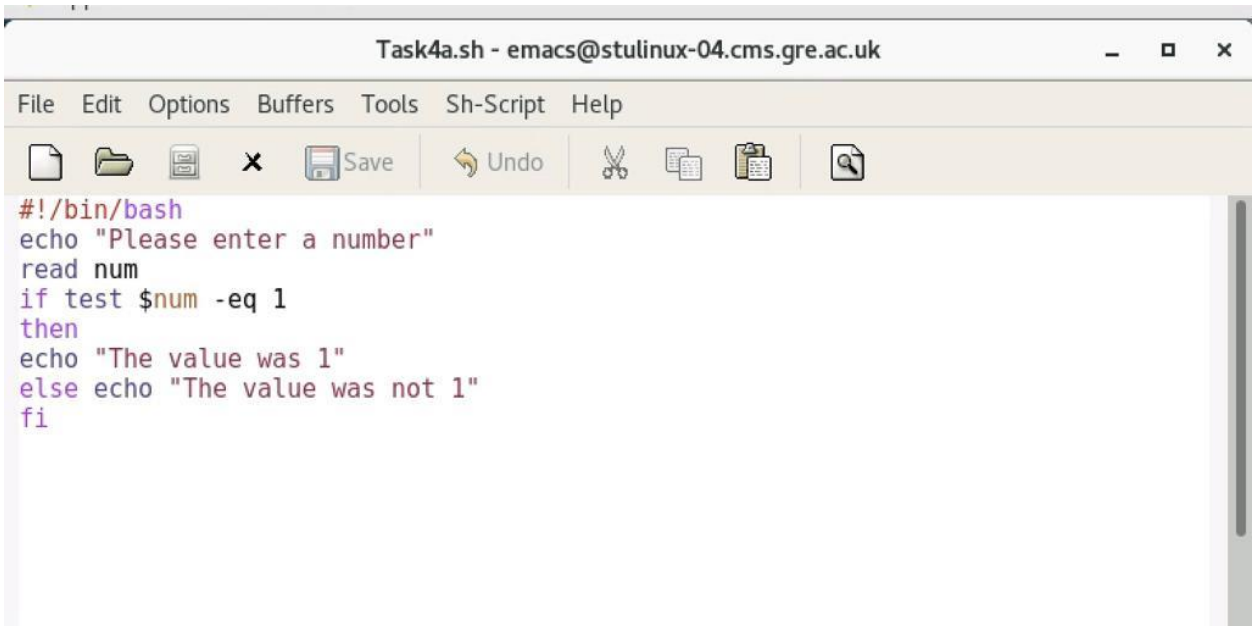
Explain the difference in file size here.

hello_asm serves more of a linking , structuring and organising function which doesn’t require much storage , where as hello.out its more functional as each character takes about 1 byte per character. On top of that it acts as a bridge between the input and out making it take more storage.

TASK 4

This was the most interesting and challenging task for our group. We learnt how to use a lot of new commands, different ways to use the previously learnt ones, how to write a mathematical expression, how to use a while loop and even how to use nested structures.

(A) Write a shell script to prompt the user for a variable, test the value of the variable, if it is 1 then write to the screen, "The value was 1", else write to the screen "The value was not 1".



Let's break this code down and analyse what it does:

#!/bin/bash is the header of the code.

All the **echo** lines will write on screen. The text can be isolated in quotation marks, not isolated at all, or **echo** can be followed by a $variable, and it will display its value. When not followed by anything else, **echo** is used to write an empty line.

read will accept input from user and store it in a variable.

if is a block that must receive a Boolean value (True/False). If the value is True, it will execute the statements within **then**. If the value is False, it will execute the statements within **else** or, if it doesn't have an **else**, it will move on to the next lines of code. To close the **if** block, we write **fi**.

test checks a statement and returns a Boolean value. In this case, it will return this value to **if**.

$ (dollar sign) is used to return the value of a variable
Ex: echo num – will write "num" on screen
echo $num – will write the value of variable num on screen

$num –eq 1 can be read as: the value of variable **num** is equal to 1.

-eq is an operand that means equal (=).

```
[cc4201c@stulinux-06 unixlab.]$ ./Task4a.sh
Please enter a number
1
The value was 1
[cc4201c@stulinux-06 unixlab.]$ ./Task4a.sh
Please enter a number
7
The value was not 1
[cc4201c@stulinux-06 unixlab.]$
```

(B) Write a shell script program that accepts three parameters and displays their average.

Let's break this code down and analyse what it does:

As the previous code, this one starts with the header. It then writes the instruction for the user. The user will have to give an input of 3 integers, all in the same line, each separated by a blank space.

sum=`expr $a + $b +$c` - this line of code assigns the variable **sum**, the sum of the variables a, b and c.

dec=`expr $sum%3` - this line of code assigns the variable **dec**, the remainder of sum divided by 3.

avg=`expr $sum/3` - this line of code assigns the variable **avg** the sum of variables, divided by 3.

echo will display both a text*(Average is:)* and the value of a variable*($avg)*

```
[cc4201c@stulinux-06 unixlab.]$ ./task4b.sh
Enter three integers with space between
5 7 8
Average is: 6
[cc4201c@stulinux-06 unixlab.]$ ./task4b.sh
```

(C) Write a shell script program that accepts a variable number of parameters and displays the average, and the number of numbers input.

task4c.sh - emacs@stulinux-04.cms.gre.ac.uk

File Edit Options Buffers Tools Sh-Script Help

Save Undo

```
#!/bin/bash
echo "How many numbers would you like to enter? (s) "
read s
i=1
sum=0
echo "enter numbers"
while [ $i -le $s ]  #-le stands for less equal <=
do
read num
sum=$((sum + num))
i=$((i + 1))
done
avg=$(echo $sum / $s | bc -l)
echo "Avarage of these numbers is $avg"
```

Let's break this code down and analyse what it does:

This code, as the previous ones, starts with a header and an instruction for the user.
The user is asked how many numbers they would like to input. This value will be stored in variable **s**.
Two variables are declared before the while loop starts: **i** and **sum**. We used variable **i** to iterate through the while loop, and **sum** to calculate the sum of the numbers the user will input.

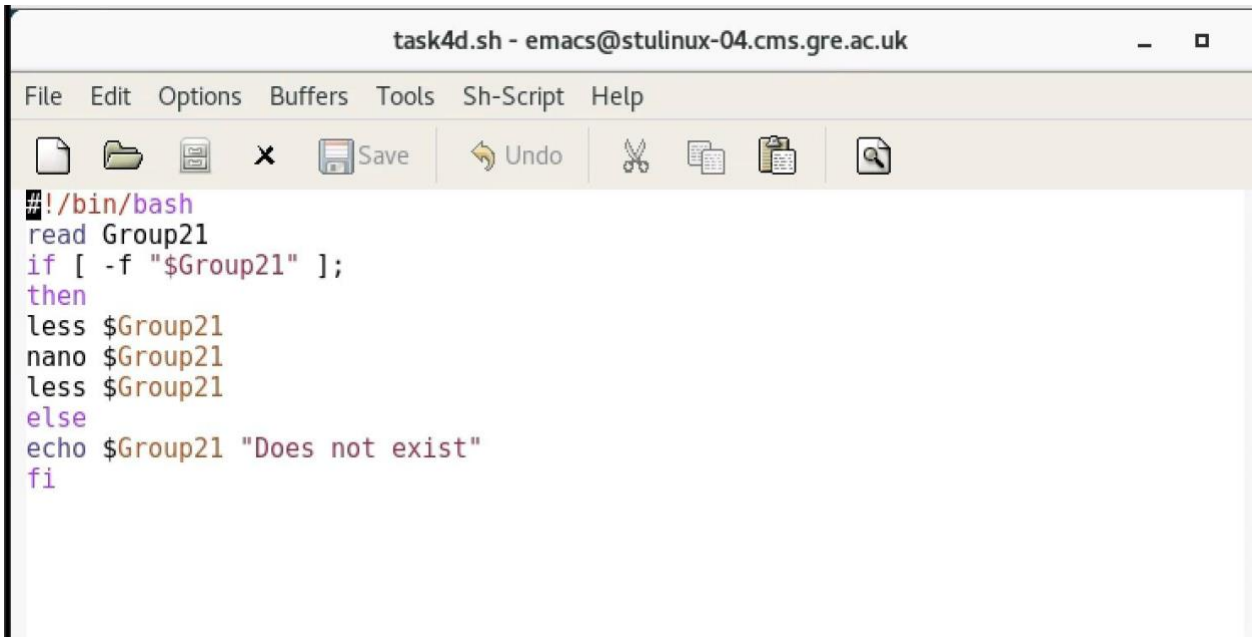
while [$i -le $s] – can be read as "while the value of variable **i** is less or equal to **s**".

The statements within the **while** loop are isolated by **do** and **done**.
With each iteration, the user will be asked to read a number, this number will be later added to the total sum
sum=$((sum+num))
and the variable **i** will increment until it's greater than variable **s** by 1
i=$((i+1))
Once this happens, the while loop will break, as [$i -le $s] is no longer true.

Then, the variable **avg** is assigned the value of **sum** divided by **s**, and the result is displayed on screen.

```
[cc4201c@stulinux-06 unixlab.]$ ./task4c.sh
How many numbers would you like to enter? (s)
5
enter numbers
4
6
8
9
7
Avarage of these numbers is 6.80000000000000000000
[cc4201c@stulinux-06 unixlab.]$
```

(D) Write a shell script program that accepts a file name from the user, displays the file using the less command, then calls the command line editor nano (command is nano filename) to allow the file to be edited, once the editor is quit, the shell script will continue to run. It should then display the file using the less command. Do not forget to test if the file exists. If it does not exist, then output an appropriate error message.



Let's break this code down and analyse what it does:

As the previous programs, this one starts with a header.

It reads an input from the user.

In the **if** statement we check whether there is a file with the name the user has just written.

If there is such a file, it will display it on screen, then allow the user to edit it, and display it again, the edited version.

If there is no such file, the user will be notified that the file they asked for does not exist.

(E) Write a shell script program to provide the following facilities to a user in the form of a menu.

- ***i. Allow the user to use the nano editor to write a poem.***
- ***ii. Run a word count on your poem using the wc command (wc poem.txt)***
- ***iii. Display the output to the screen and display a page at a time using more or less, ask the user which one they want to use.***
- ***iv. Sort the output using the sort command and put the results into a file, then display the output of the sorted file using cat or more, ask the user which one they want to use.***
- ***v. Exit***

task4e.sh - emacs@stulinux-04.cms.gre.ac.uk

File Edit Options Buffers Tools Sh-Script Help

Save Undo

```
#!/bin/bash
stop=0
while test $stop = 0
do
    echo
    echo
    echo Menu Program
    echo
    echo 1     : Edit the poem
    echo 2     : Do the Word Count for the poem
    echo 3     : Display poem
    echo 4     : Sort
    echo 5     : Exit
    echo
    echo 'Please enter your choice? '
    read reply
    case $reply in
       "1"            ) nano poem.txt  ;;

       "2"            ) wc poem.txt    ;;

       "3"            ) echo 1 : more
                        echo 2 : less
                        read reply1
                        case $reply1 in
                           "1") more poem.txt  ;;
                           "2") less poem.txt  ;;
                            * )  echo illegal choice  ;;
                        esac;;

       "4"             )sort poem.txt > poemtemp.txt
                        echo 1 : cat
                        echo 2 : more
                        read reply2
                        case $reply2 in
                           "1") cat poemtemp.txt   ;;
                           "2") more poemtemp.txt  ;;
                            * )  echo illegal choice  ;;
                        esac ;;

       "5"            ) stop=1 ;;

       *              ) echo illegal choice ;;
  esac
```

First, we declare a variable called **stop** and assign it the value 0. The **while** loop uses the test command to check whether variable **stop** is equal to 0. As long as this is the case, the test command will return **True** (Boolean value) and the menu will keep running. Once the user chooses option 5, which is exit, variable **stop** will be reassigned the value of 1. When the while loop will try to run again, the test command will return **False** (Boolean value), meaning that **stop** is no longer equal to 0, and the menu will stop.

Using multiple **echo** lines, we have displayed a menu for the user to see. The user will choose one of the options listed, and the variable **reply** will store the user's choice.

What is **case** used for? **case** is usually used to substitute multiple **if**s and **else if**s, as it is more optimised and uses less memory. However, if the if statement is a short one, it is more reasonable to use if, and not case.

The **case** block is opened. If **reply** is a number between 1 and 4, a statement/set of statements will be executed. If **reply** is 5, the menu stops, as explained before. If **reply** is any other character, the user will be notified that his choice is illegal, and the menu will open again for the user to choose another option.

This is what the menu looks like:

```
[cc4201c@stulinux-06 unixlab.]$ ./task4e.sh


Menu Program

1 : Edit the poem
2 : Do the Word Count for the poem
3 : Display poem
4 : Sort
5 : Exit

Please enter your choice?
```

This is what happens if an illegal option is chosen:

```
Menu Program

1 : Edit the poem
2 : Do the Word Count for the poem
3 : Display poem
4 : Sort
5 : Exit

Please enter your choice?
GROUP 59
illegal choice
```

And the menu will display again, so the user can choose another option:

```
[cc4201c@stulinux-06 unixlab.]$ ./task4e.sh


Menu Program

1 : Edit the poem
2 : Do the Word Count for the poem
3 : Display poem
4 : Sort
5 : Exit

Please enter your choice?
```

Case "1":

The user will get to edit the **poem.txt** in right in the terminal. We achieved it using **nano** command.

```
Hello Group21
And a Heaven in a Wild Flower,
Hold Infinity in the palm of your hand
And Eternity in an hour.

A Robin Redbreast in a Cage
Puts all Heaven in a Rage.
A dove house fill'd with doves and pigeons
Shudders Hell thro' all its regions.
A Dog starv'd at his Master's Gate
Predicts the ruin of the State.
A Horse misus'd upon the Road
Calls to Heaven for Human blood.
Each outcry of the hunted Hare
A fiber from the Brain does tear.

He who shall train the Horse to War
Shall never pass the Polar Bar.
The Beggar's Dog and Widow's Cat,
Feed them and thou wilt grow fat.
The Gnat that sings his Summer song
Poison gets from Slander's tongue.
The poison of the Snake and Newt
                                   [ Read 46 lines ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

After each case, a double semi-colon should be used, to show where a case ends and a new one starts.

Case "2":

The user will be able to check the word count of the **poem.txt** file. This statement will return the number of lines, words and characters in the file.

```
Menu Program

1 : Edit the poem
2 : Do the Word Count for the poem
3 : Display poem
4 : Sort
5 : Exit

Please enter your choice?
2
  46  211 1109 poem.txt
```

Case "3":

The user chose to display the **poem.txt** file. Using another **case** block (nested case), we asked the user how they would like the **poem.txt** file to be displayed.

```
Please enter your choice?
4
1 : cat
2 : more
```

If user chooses 1, poem.txt will be displayed using **more** command.

If user chooses 2, poem.txt will be displayed using **less** command.

If user's input is anything but 1 or 2, the user will be notified that their choice is illegal.

After closing the nested **case**, using **esac** key-word, we again need a double semi-colon, otherwise there will be an error, when executing the program.

Case "4":

sort poem.txt > poemtemp.txt – is a statement that will sort the **poem.txt**, using redirection(**>**) it will store the result in **poemtemp.txt**. At this point, user will not see that the file has been sorted.

```
Menu Program

1 : Edit the poem
2 : Do the Word Count for the poem
3 : Display poem
4 : Sort
5 : Exit

Please enter your choice?
4
1 : cat
2 : more
```

User is now given a choice, of how they would like the sorted text file to be displayed: using **cat** or **more** commands. For that, we used another nested **case**. Again, if the choice is anything other than 1 or 2, the user will be notified that their choice is illegal.

```
stulinux-login:6000 - Remote Desktop Connection
And a Heaven in a Wild Flower,
Hold Infinity in the palm of your hand
And Eternity in an hour.

A Robin Redbreast in a Cage
Puts all Heaven in a Rage.
A dove house fill'd with doves and pigeons
Shudders Hell thro' all its regions.
A Dog starv'd at his Master's Gate
Predicts the ruin of the State.
A Horse misus'd upon the Road
Calls to Heaven for Human blood.
Each outcry of the hunted Hare
A fiber from the Brain does tear.

He who shall train the Horse to War
Shall never pass the Polar Bar.
The Beggar's Dog and Widow's Cat,
Feed them and thou wilt grow fat.
The Gnat that sings his Summer song
Poison gets from Slander's tongue.
The poison of the Snake and Newt
Is the sweat of Envy's Foot.

A truth that's told with bad intent
Beats all the Lies you can invent.
poem.txt
```

Case "5":

User chose to exit the menu. Variable **stop** is assigned value 1, the loop is now broken and the menu is no longer running.

```
Menu Program

1 : Edit the poem
2 : Do the Word Count for the poem
3 : Display poem
4 : Sort
5 : Exit

Please enter your choice?
5
[cc4201c@stulinux-06 unixlab.]$
```

REFERENCES;

Cs.lmu.edu. 2021. *x86assembly*. [online] Available at: <https://cs.lmu.edu/~ray/notes/x86assembly/> [Accessed 3 October 2021].

Linuxhint.com. 2021. *30 Bash Script Examples*. [online] Available at: <https://linuxhint.com/30_bash_script_examples/#t6> [Accessed 5 October 2021].

Log2base2.com. 2021. *Shell program to find average of n numbers*. [online] Available at: <https://www.log2base2.com/shell-script-examples/loop/shell-program-to-find-average-of-n-numbers.html> [Accessed 2 October 2021].