



BASH SCRIPTING

OPERATING SYSTEM



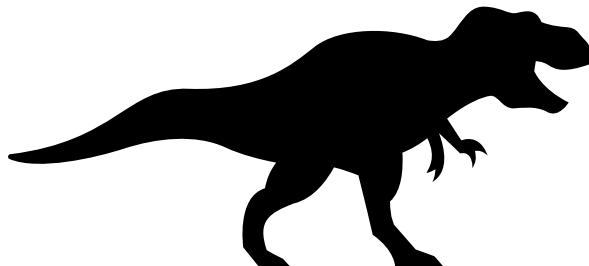
COLLABORATORS NAME & ID :
MARYAM ALIKARAMI 9731045
MAHAN AHMADVAND 9731071



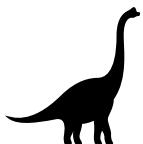


BASH SCRIPTING

OPERATING SYSTEM



COLLABORATORS NAME & ID :
MARYAM ALIKARAMI 9731045
MAHAN AHMADVAND 9731071



OPERATING SYSTEM LABORATORY

BASH SCRIPTING



INSTRUCTOR :

ARMAN GHEISARI

COLLABORATORS :

MARYAM ALIKARAMI

MAHAN AHMADVAN,

LEAD-IN

ابتدا می خواهیم با مفهومی به نام Bash Scripting آشنا شویم.

Bash یکی از shell های لینوکس است که به عنوان یک مفسر مورد استفاده قرار می گیرد و دستورات را از ما می گیرد و پردازش می کند و خروجی را به ما تحویل می دهد حال Scripting به این معناست که ما مجموعه ای از دستورات Bash را در یک فایل بنویسیم و آنها را بصورت دستی با هم اجرا کنیم، یعنی بجای آنکه تک به تک دستورات را وارد ترمینال کنیم، آنها را به صورت دسته ای اجرا کنیم.

زبان های برنامه نویسی به دو دسته‌ی کامپایلری و مفسری تقسیم می شوند که Bash یک زبان مفسری است. حال می خواهیم که با زبان های مفسری و کامپایلری کمی بیشتر آشنا شویم.

مفسرها برای سیستم عامل های مختلفی وجود دارند و وظیفه‌ی مفسر ها این است که هر خط برنامه را می خوانند و اجرا می کنند، مشکلی که مفسرها دارند این است که فقط می توانند برخی از ویژگی های سیستم عامل را ارائه دهند.

مفسرها زمان کمی را برای تحلیل source code صرف می کنند، درنتیجه زمان اجرای کد در مجموع کند تر از compiler ها است.

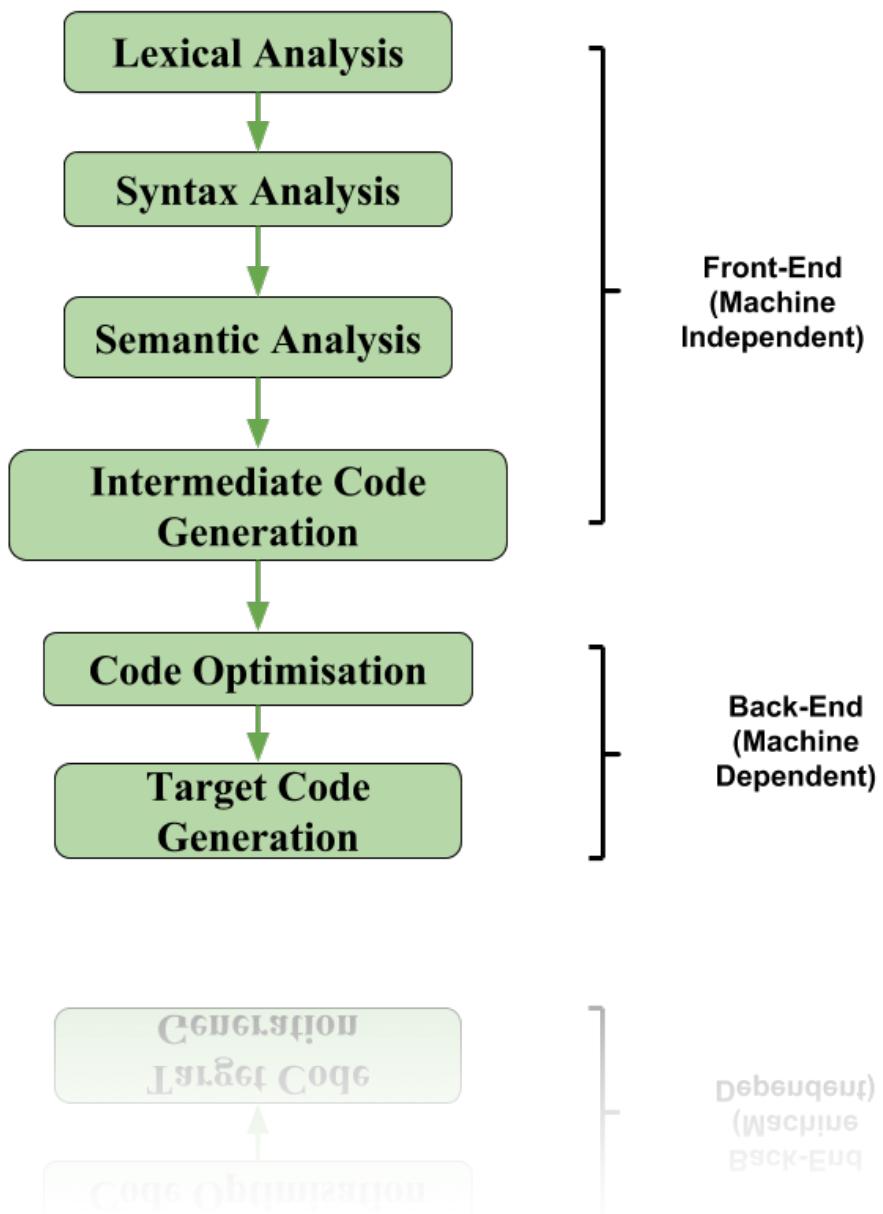
به دلیل اینکه مفسرها **intermediate object code** تولید نمی کنند، از جهت حافظه بهینه تر هستند (بهینه تر هستند نسبت به **compiler** ها).

عمل عیب یابی (debugging) در مفسرها راحت تر است، زیرا هر خطی را که می خوانند اجرا می کنند و می توان فهیم که ارور در چه خطی رخ داده است.

زبان هایی که از شیوه **compile** شدن استفاده می کنند، به دلیل تولید **intermediate object code** از نظر حافظه نسبت به مفسرها بهینه نیستند.

زبان های کامپایلری زمان زیادتری را برای تحلیل **source code** صرف می کنند، درنتیجه زمان اجرای کد در مجموع سریع تر از مفسر ها است.

در زبان های کامپایلری، توسعه دهنده از API ها استفاده می کند و کامپایلر کد ها را به زبان ماشین تبدیل می کند. کامپایلر ها دو بخش **frontend** و **backend** دارند که قسمت **backend** وابسته به سخت افزار است و اگر سیستم تغییر کند باید دوباره برای سیستم جدید **backend** نوشته شود، اما **frontend** وابسته به سخت افزار نیست و اگر سیستم تغییر کند نیازی به تغییر آن نیست.



همچنین می خواهیم کمی در مورد ماشین مجازی جاوا نیز توضیح دهیم.

ماشین مجازی جاوا برای سیستم عامل های مختلفی توسعه پیدا کرده است و می توان کدهای جاوا را ببروی سیستم عامل های مختلفی اجرا کرد.

در واقع زبان جاوا به این صورت عمل می کند که `Javac` (کامپایلر جاوا) کد جاوا را به `Bytecode` کامپایل می کند و سپس `Java Virtual Machine` (JVM) `Bytecode` را بحسب آمده را به زبان ماشین تفسیر می کند، در واقع عمل `Just in time` تفسیر `Bytecode` به زبان ماشین به صورت `Just in time` صورت می گیرد.

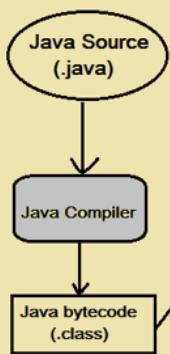
در واقع JVM مفسری است که برای سیستم عامل های مختلفی نوشته شده است و از این جهت یک مزیت است که بر هر سیستمی که JVM برای آن نوشته شده است به اصطلاح `port` می شود.

در واقع جاوا از هردو روش مفسری و کامپایلر استفاده می کند.

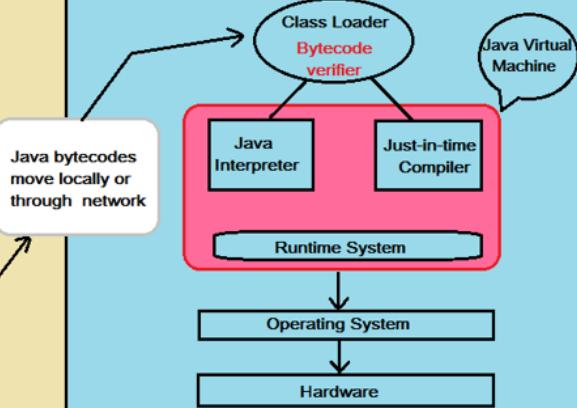
خوبی زبان جاوا این است که `java bytecode` را یکبار می نویسیم و ببروی هر سیستمی که JVM برای آن نوشته شده باشد می توانیم آن را `Run` کنیم یعنی `Platform Independent` است.

زبان جاوا بسیار به JVM وابسته است و در صورتی که JVM `crash` کند عملاً کل برنامه دچار مشکل می شود. JVM نیاز به حافظه دارد و خود یک برنامه است و مقدار قابل توجه ای از حافظه را اشغال می کند.

Compile-time Environment



Run-time Environment



QUESTION 1

دست نوشته (اسکریپتی) بنویسید که دو عددی که به صورت آرگومان به آن داده شده را
الف - با هم جمع کند و نتیجه را اعلام کند.
ب - عدد بزرگتر را نمایش دهد.
ج - اگر کاربر در وارد کردن ورودی ها اشتباه کرده بود
راهنمای مناسب چاپ کند.

ابتدا به دایرکتوری میزکار می رویم و یک فایل به اسم Q1.sh با استفاده از دستور gedit می سازیم.
حال می خواهیم از مفهوم اسکریپت استفاده کنیم، یعنی دستورات خود را در یک فایل بنویسیم و آن را اجرا کنیم.
حال کد موارد خواسته شده در سوال را پیاده سازی کرده ایم
و در این فایل کپی می کنیم، مطابق شکل :

The screenshot shows a Linux desktop environment with a dark theme. On the left, there's a file manager window showing icons for 'mahan', 'M', 'Trash', 'Q', and 'Q1.sh'. In the center, there's a terminal window titled 'mahan@mahan-VirtualBox: ~\$' displaying the command 'ls'. To the right, there's a large file editor window titled 'Q1.sh' showing the script code. The script is a shell script named Q1.sh that takes two arguments, compares them, and prints the sum or the greater value.

```
#!/bin/bash
# Checking the number of arguments
if [[ $1 == "" ]]; then
    echo "Number of arguments must be 2"
    exit 1
fi

# Checking if the arguments are numeric
for i in $1 $2; do
    if ! [[ $i =~ [0-9]+([.][0-9]+)? ]]; then
        echo "$i is not a numeric value"
        exit 1
    fi
done

# Adding sum of two arguments
echo "Sum of two arguments is: $(($1 + $2))"

# Comparing arguments
if [[ $1 > $2 ]]; then
    echo "$1 is greater than $2"
elif [[ $1 == $2 ]]; then
    echo "$1 and $2 are equal"
else
    echo "$2 is greater than $1"
fi
```

حال به جزئیات کد می پردازیم :

```
#!/bin/bash

# Checking the number of arguments
if [[ $# -ne 2 ]]; then
    echo "Number of arguments must be 2"
    exit 1
fi

# Checking if the arguments are numeric
regex='^[-]?[0-9]+$'
for i in "$@"; do
    if [[ ! $i =~ $regex ]]; then
        echo "\"$i\" is not a numeric value"
        exit 1
    fi
done

# Printing sum of two arguments
echo "Sum of two arguments is: $(( $1 + $2 ))"

# Comparing arguments
if [[ $1 -gt $2 ]]; then
    echo "$1 is greater than $2"

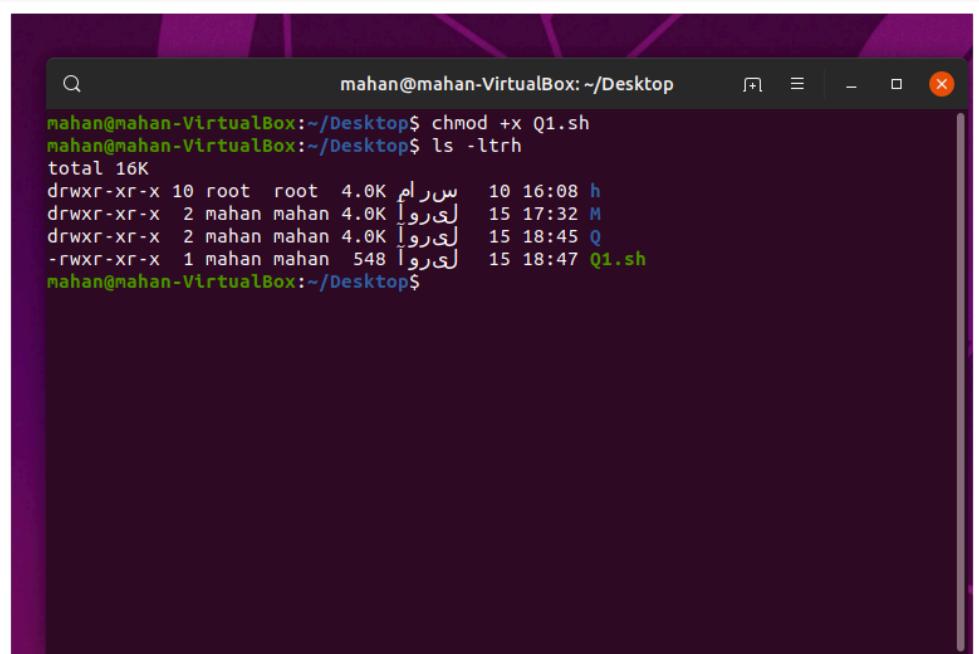
elif [[ $1 -eq $2 ]]; then
    echo "$1 and $2 are equal"

else
    echo "$2 is greater than $1"
fi
```

خط اول کد `#!/bin/bash` به خط شبنگ معروف است، یعنی اول هر اسکریپتی که می خواهیم اجرا کنیم، تفسیر کننده ی آن فایل را اول خط می نویسیم یعنی لینوکس می فهمد که باید این

فایل را با استفاده از برنامه ای که در آدرس `#!/bin/bash` قرار دارد بخواند.

حال با استفاده از دستور `chmod +x Q1.sh` به این فایل قابلیت اجرایی نیز می دهیم و با استفاده از دستور `ls -ltrh` چک می کنیم که این کار به درستی انجام شده باشد :



A screenshot of a terminal window titled "mahan@mahan-VirtualBox: ~/Desktop". The terminal shows the following command and its output:

```
mahan@mahan-VirtualBox:~/Desktop$ chmod +x Q1.sh
mahan@mahan-VirtualBox:~/Desktop$ ls -ltrh
total 16K
drwxr-xr-x 10 root root 4.0K سرگام 10 16:08 h
drwxr-xr-x  2 mahan mahan 4.0K لیورو 15 17:32 M
drwxr-xr-x  2 mahan mahan 4.0K لیورو 15 18:45 Q
-rwxr-xr-x  1 mahan mahan 548 لیورو 15 18:47 Q1.sh
mahan@mahan-VirtualBox:~/Desktop$
```

حال می خواهیم تمام سناریو های پیاده سازی شده در کد بالا را بررسی کنیم :

```
mahan@mahan-VirtualBox:~/Desktop$ ./Q1.sh 5
Number of arguments must be 2
mahan@mahan-VirtualBox:~/Desktop$ ./Q1.sh 5 b
'b' is not a numeric value
mahan@mahan-VirtualBox:~/Desktop$ ./Q1.sh -8 2
Sum of two arguments is: -6
2 is greater than -8
mahan@mahan-VirtualBox:~/Desktop$ ./Q1.sh 8 9
Sum of two arguments is: 17
9 is greater than 8
mahan@mahan-VirtualBox:~/Desktop$ ./Q1.sh 4 4
Sum of two arguments is: 8
4 and 4 are equal
mahan@mahan-VirtualBox:~/Desktop$ ./Q1.sh 5 4
Sum of two arguments is: 9
5 is greater than 4
mahan@mahan-VirtualBox:~/Desktop$
```

در خروجی های بالا مشاهده می کنیم با دریافت دو آرگومان ورودی جمع آن ها و عدد بزرگتر چاپ شده است، همچنین اگر تعداد آرگومان ها برابر ۲ نباشد و یا آرگومان ها به شکل عدد صحیح نباشند مطابق شکل خطای چاپ شده است.

تحلیل نکات مهم کد :

ابتدا چک کرده ایم(با استفاده از if) که اگر تعداد آرگومان های ورودی برابر ۲ نبود با استفاده از دستور echo پیامی به کاربر نمایش داده شود و سپس با استفاده از ۱ خارج شدیم.

توجه شود که برای شرط if نوشته ایم `2 -ne $#` این به این معنی است که اگر تعداد آرگمان ها برابر ۲ نبود(زیرا `-ne` داریم)، `status code` صفر برگردانده می شود در نتیجه کد if اجرا می شود، همچنین متغیر `$#` تعداد آرگمان های ورودی را به ما می دهد.

در قسمت بعد کد چک کرده ایم که آیا آرگومان های ورودی عدد صحیح هستند یا خیر.

این کار را با استفاده از مقایسه کردن رجکس `?[-]+` انجام داده ایم به این صورت که با استفاده از `[@]}` که به تمام آرگومان های ورودی ارجاع می دهد و با استفاده از `for` چک کرده ایم که اگر هر کدام از آرگومان های ورودی با رجکس ما همخوانی نداشت آنگاه پیامی به کاربر چاپ کرده ایم و با استفاده از `exit 1` خارج شده ایم.

در قسمت بعدی کد با استفاده از `$1 + $2` که `$1` به آرگومان اول و `$2` به آرگومان دوم اشاره دارد این دو مقدار ورودی را جمع کرده ایم و با استفاده از دستور `echo` مقدار آن را چاپ کرده ایم.

در قسمت آخر کد با استفاده از `if` و `elif` حالت های مختلف بزرگتری و کوچکتری و برابری را چک کرده ایم که نیاز به توضیح ندارد:))).

QUESTION 2

ماشین حسابی با استفاده از `case` طراحی کنید.
ابتدا به دایرکتوری میزکار می رویم و یک فایل به اسم `Q2.sh` با استفاده از دستور `gedit Q2.sh` می سازیم
حال می خواهیم از مفهوم اسکریپت استفاده کنیم، یعنی دستورات خود را در یک فایل بنویسیم و آن را اجرا کنیم.
حال کد موارد خواسته شده در سوال را پیاده سازی کرده ایم
و در این فایل کپی می کنیم، مطابق شکل :

The screenshot shows a Linux desktop environment with a purple gradient background. On the left, there's a file manager window showing files like 'Q2.sh' and 'Q1.sh'. In the center, there's a terminal window titled 'Q' with the command: 'mahard@mahar:~\$ cd Desktop'. To its right, there's another terminal window titled 'Q' with the command: 'mahard@mahar:~/Desktop\$ gedit Q2.sh'. Below these windows, a code editor window titled 'Q2.sh - Untitled' is open, displaying the following shell script:

```
#!/bin/bash
read -p "Enter first operand: " op1
read -p "Enter operand: " operand
read -p "Enter second operand: " op2
echo -n "Result: "
case $operand in
    +)
        expr "$op1" + "$op2"
    ;;
    -)
        expr "$op1" - "$op2"
    ;;
    *)
        expr "$op1" * "$op2"
    ;;
    /)
        expr "$op1" / "$op2"
    ;;
esac
```

حال به جزئیات کد می پردازیم :

```
#!/bin/bash

read -p "Enter first operator: " op1
read -p "Enter operand: " operand
read -p "Enter second operator: " op2

echo -n "Result: "
case $operand in
    +)
        expr "$op1" + "$op2";;
    -)
        expr "$op1" - "$op2";;
    \*)
        expr "$op1" \* "$op2";;
    /)
        expr "$op1" / "$op2";;
esac
```

حال با استفاده از دستور chmod +x Q2.sh به این فایل قابلیت اجرایی نیز می دهیم و با استفاده از دستور ls -l rh چک می کنیم که این کار به درستی انجام شده باشد :

```
mahan@mahan-VirtualBox:~/Desktop$ chmod +x Q2.sh
mahan@mahan-VirtualBox:~/Desktop$ ls -ltrh
total 20K
drwxr-xr-x 10 root root 4.0K سرام 10 16:08 h
drwxr-xr-x 2 mahan mahan 4.0K لیرو 15 18:45 Q
-rwxr-xr-x 1 mahan mahan 548 لیرو 15 18:47 Q1.sh
drwxr-xr-x 2 mahan mahan 4.0K لیرو 15 19:26 M
-rwxr-xr-x 1 mahan mahan 283 لیرو 15 19:28 Q2.sh
mahan@mahan-VirtualBox:~/Desktop$
```

حال می خواهیم تمام سناریو های پیاده سازی شده در کد بالا را بررسی کنیم :

```
mahan@mahan-VirtualBox:~/Desktop$ ./Q2.sh
Enter first operator: 2
Enter operand: +
Enter second operator: 3
Result: 5
mahan@mahan-VirtualBox:~/Desktop$ ./Q2.sh
Enter first operator: 9
Enter operand: -
Enter second operator: -7
Result: 16
mahan@mahan-VirtualBox:~/Desktop$ ./Q2.sh
Enter first operator: 4
Enter operand: *
Enter second operator: 8
Result: 32
mahan@mahan-VirtualBox:~/Desktop$ ./Q2.sh
Enter first operator: 64
Enter operand: /
Enter second operator: 8
Result: 8
mahan@mahan-VirtualBox:~/Desktop$
```

همانطور که مشاهده می کنیم عملیات های جمع و تفریق و ضرب و تقسیم به درستی پیاده سازی شده اند.
تحلیل نکات مهم کد :

همانطور که مشاهده می کنید با استفاده از دستور `read`-`p` از `prompt` می آید می توان یک رشته را پیش از ورودی گرفتن به عنوان راهنمای کاربر چاپ کرد و می توان از کاربر ورودی خواند و در متغیر ذخیره کرد که با این کار `op1` و `op2` را از کاربر گرفته ایم و با استفاده از دستور `case` چک کرده ایم که هر کدام از `operand` های `/`, `*`, `,`, `-`, `+` بود، عملیات مربوط به هر کدام انجام شود.

QUESTION 3

برنامه ای بنویسید که به طور متوالی از کاربر عدد دریافت کند و عددی چاپ کند که ترتیب ارقامش معکوس باشد. مثلا ۵۶۷ را به صورت ۷۶۵ چاپ کند. سپس جمع ارقام آن را چاپ کند. ابتدا به دایرکتوری میزکار می رویم و یک فایل به اسم Q3.sh با استفاده از دستور gedit Q3.sh می سازیم. حال می خواهیم از مفهوم اسکریپت استفاده کنیم، یعنی دستورات خود را در یک فایل بنویسیم و آن را اجرا کنیم. حال کد موارد خواسته شده در سوال را پیاده سازی کرده ایم و در این فایل کپی می کنیم، مطابق شکل :

The screenshot shows a Linux desktop environment with a dark theme. On the left, there is a file manager window showing icons for 'mahan', 'M', 'Q3.sh', 'trash', and 'h'. In the center, there is a terminal window titled 'Q3.sh' containing the following shell script:

```
#!/bin/bash
function sum() {
    local number=$1
    local sum=0
    while [[ $number -gt 0 ]]; do
        ((sum+=${number%$((number-1))}))
        ((number/=10))
    done
    echo $sum
}
read -p "Enter a number: " number
sum $number
echo "Sum of digits is $sum"
echo "Reverse Stecho Snumber | rev"
read -p "Enter a number: " number
done
```

To the right of the terminal is another terminal window with the command 'mahan@mahan-VirtualBox:~/Desktop\$ cd Desktop' and 'mahan@mahan-VirtualBox:~/Desktop\$ gedit Q3.sh' entered.

حال به جزئیات کد می پردازیم :

```
#!/bin/bash

function sum() {
    local number=$1
    local sumOfDigits=0
    while [[ $number -ne 0 ]]; do
        ((sumOfDigits+=number%10))
        ((number/=10))
    done
    echo $sumOfDigits
}

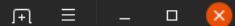
read -p "Enter a number: " number
while [[ "$number" -ne 0 ]]; do
    echo "Sum of digits is $(sum $number)"
    echo "Reverse ${echo $number | rev}"

    read -p "Enter a number: " number
done
```

حال با استفاده از دستور chmod +x Q3.sh به این فایل قابلیت اجرایی نیز می دهیم و با استفاده از دستور ls -lthr چک می کنیم که این کار به درستی انجام شده باشد :



mahan@mahan-VirtualBox: ~/Desktop



```
mahan@mahan-VirtualBox:~/Desktop$ chmod +x Q3.sh
mahan@mahan-VirtualBox:~/Desktop$ ls -ltrh
total 24K
drwxr-xr-x 10 root  root  4.0K سر ام 10 16:08 h
drwxr-xr-x  2 mahan mahan 4.0K لیرو 15 18:45 Q
-rwxr-xr-x  1 mahan mahan 548 لیرو 15 18:47 Q1.sh
-rwxr-xr-x  1 mahan mahan 283 لیرو 15 19:28 Q2.sh
drwxr-xr-x  2 mahan mahan 4.0K لیرو 15 21:51 M
-rwxr-xr-x  1 mahan mahan 359 لیرو 15 21:51 Q3.sh
mahan@mahan-VirtualBox:~/Desktop$
```

حال می خواهیم تمام سناریو های پیاده سازی شده در کد
بالا را بررسی کنیم :

```
mahan@mahan-VirtualBox:~/Desktop$ ./Q3.sh
```

```
Enter a number: 123
Sum of digits is 6
Reverse 321
Enter a number: 546
Sum of digits is 15
Reverse 645
Enter a number: 809
Sum of digits is 17
Reverse 908
Enter a number: 333
Sum of digits is 9
Reverse 333
Enter a number: 234
Sum of digits is 9
Reverse 432
Enter a number: 984
Sum of digits is 21
Reverse 489
Enter a number: ■
```

همانطور که می بینیم عملیات reverse و مجموع ارقام به درستی انجام شده است.
تحلیل نکات مهم کد :

برای جمع کردن مجموع ارقام تابع sum را نوشته ایم که دو متغیر local، number که عدد ورودی کاربر است و sumOfDigits که مقدار مجموع ارقام را در خود نگه داری می کند، حال با استفاده از while و شرط \$number -ne 0 باقیمانده بار با استفاده از ((sumOfDigits+=number%10)) عدد ورودی را برابر ۱۰ حساب کرده و در sumOfDigits می ریزیم و سپس خارج قسمت number بر ۱۰ را حساب کرده و در number می ریزیم و به این ترتیب این کار را انجام می دهیم تا زمانی که number صفر شود آنگاه مجموع ارقام عدد ورودی کاربر را نشان خواهد داد.

حال با استفاده از یک while دیگر به تا زمانی که کاربر عدد ۰ را وارد نکرده از کاربر ورودی می گیریم، همچنین با استفاده از دستور \$number | rev عدد ورودی کاربر را reverse کرده و با استفاده از دستور echo آن را نمایش داده ایم.

QUESTION 4

برنامه ای بنویسید که در هنگام اجرا دو عدد x و y و اسم یک فایل را دریافت کند و در خروجی خط x ام تا y ام فایل مذکور را نمایش دهد.

ابتدا به دایرکتوری میزکار می رویم و یک فایل به اسم Q4.sh با استفاده از دستور gedit Q4.sh می سازیم.

حال می خواهیم از مفهوم اسکریپت استفاده کنیم، یعنی دستورات خود را در یک فایل بنویسیم و آن را اجرا کنیم.

حال کد موارد خواسته شده در سوال را پیاده سازی کرده ایم و در این فایل کپی می کنیم، مطابق شکل :

```
#!/bin/bash
read -p "Enter file name: " file
read -p "Enter the starting line number: " x
read -p "Enter the stopping line number: " y
head -n $y $file | tail -n ${($y - $x + 1)}
```

```
mahan@mahan-VirtualBox:~/Desktop$ gedit Q4.sh
mahan@mahan-VirtualBox:~/Desktop$
```

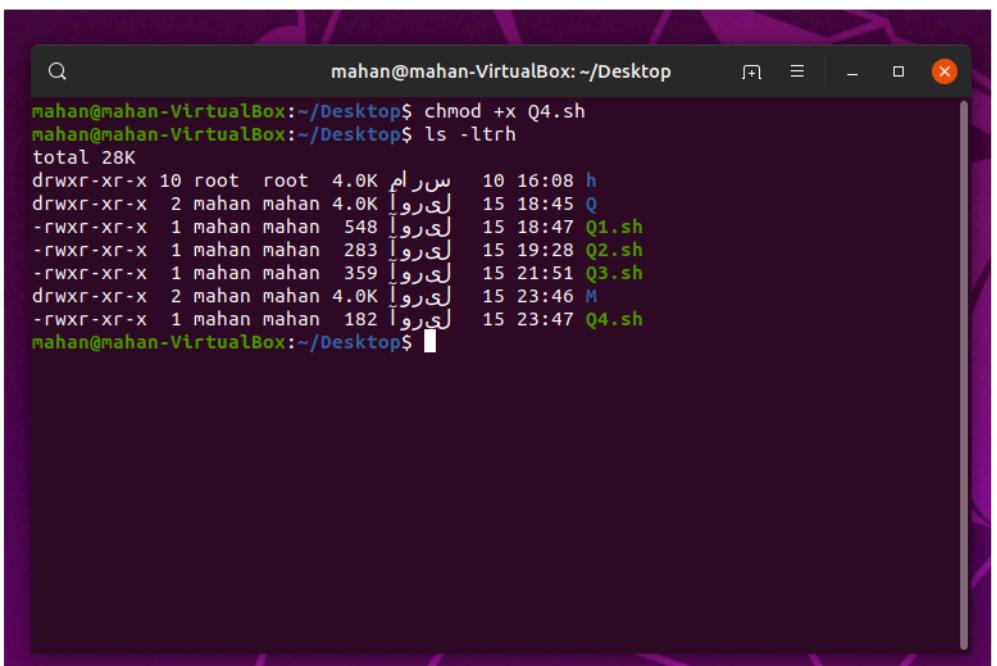
حال به جزئیات کد می پردازیم :

```
#!/bin/bash

read -p "Enter file name: " file
read -p "Enter the starting line number: " x
read -p "Enter the stopping line number: " y

head -n $y $file | tail -n $((y - x + 1))
```

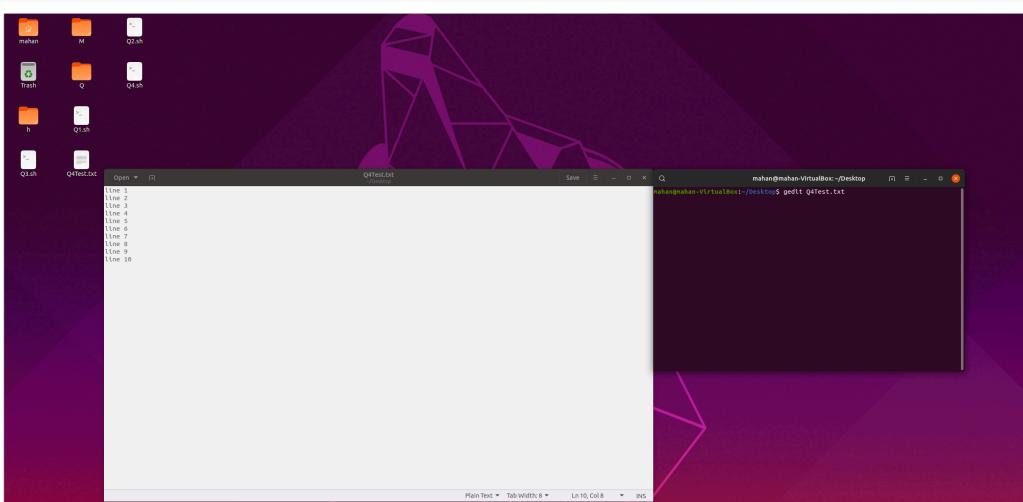
حال با استفاده از دستور chmod +x Q4.sh به این فایل قابلیت اجرایی نیز می دهیم و با استفاده از دستور ls -lthr چک می کنیم که این کار به درستی انجام شده باشد :



mahan@mahan-VirtualBox:~/Desktop\$ chmod +x Q4.sh
mahan@mahan-VirtualBox:~/Desktop\$ ls -lthr
total 28K
drwxr-xr-x 10 root root 4.0K سر ام 10 16:08 h
drwxr-xr-x 2 mahan mahan 4.0K لیرو 15 18:45 Q
-rwxr-xr-x 1 mahan mahan 548 لیرو 15 18:47 Q1.sh
-rwxr-xr-x 1 mahan mahan 283 لیرو 15 19:28 Q2.sh
-rwxr-xr-x 1 mahan mahan 359 لیرو 15 21:51 Q3.sh
drwxr-xr-x 2 mahan mahan 4.0K لیرو 15 23:46 M
-rwxr-xr-x 1 mahan mahan 182 لیرو 15 23:47 Q4.sh
mahan@mahan-VirtualBox:~/Desktop\$

حال می خواهیم تمام سناریو های پیاده سازی شده در کد بالا را بررسی کنیم :

برای این کار یک فایل به اسم Q4Test.txt ایجاد می کنیم و در مقادیری می نویسیم، سپس با استفاده از دستور cat Q4Test.txt مقادیر هر خط فایل Q4Test.txt را چاپ می کنیم تا دید بهتری نسبت به محتوای فایل بگیریم و سپس کد را تست می کنیم :



```
mahan@mahan-VirtualBox:~/Desktop$ cat Q4Test.txt
line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
line 10
mahan@mahan-VirtualBox:~/Desktop$ ./Q4.sh
Enter file name: Q4Test.txt
Enter the starting line number: 5
Enter the stopping line number: 9
line 5
line 6
line 7
line 8
line 9
mahan@mahan-VirtualBox:~/Desktop$
```

می بینیم که تست با موفقیت انجام شد و کد پیاده سازی شده درست کار می کند.
تحلیل نکات مهم کد :

از دستور

انجام این کار استفاده کرده ایم، `-n` به این دلیل است که، اگر ما از سوئیچ `-n` استفاده نکنیم، حالت دیفالت تعداد خطوطی که دستور `head` میخواند ۱۰ می باشد اما با سوئیچ `-n` می توانیم این تعداد را کم و زیاد کنیم، همچنین این امر برای `tail` نیز صادق است و حالت دیفالت `tail` نیز ۱۰ است.

حال باید به این نکته توجه کنیم که اگر عدد جلوی `head -n` مثبت باشد به آن اندازه تعداد خط، از اول فایل می خواند، اما

اگر این عدد منفی باشد به اندازه این عدد از آخر فایل حذف می کند و بقیه را به ما نشان می دهد.

در مورد `tail -n`، اگر عدد بعد از این دستور مثبت باشد به اندازه این عدد مثبت تعداد خط از آخر فایل می خواند، اما اگر منفی باشد به اندازه این عدد از اول فایل حذف کرده و بقیه را می خواند.

در نتیجه باید با این دو مقدار بازی کرده و زمانی که میخواهیم از یک خط مشخص از فایل تا یک خط مشخص دیگر بخوانیم از ترکیب این دو دستور استفاده می کنیم به این صورت که به اندازه `u` خط از اول می خوانیم(این کار را `head` انجام می دهد) و سپس به انداز `x+1-y` از این مقدار خوانده شده را می بریم(یعنی `tail` این مقدار را `cut` می کند).

QUESTION 5

Bubble sort را پیاده سازی کنید.

ابتدا به دایرکتوری میزکار می رویم و یک فایل به اسم Q5.sh با استفاده از دستور gedit Q5.sh می سازیم.
حال می خواهیم از مفهوم اسکریپت استفاده کنیم، یعنی دستورات خود را در یک فایل بنویسیم و آن را اجرا کنیم.
حال کد موارد خواسته شده در سوال را پیاده سازی کرده ایم و در این فایل کپی می کنیم، مطابق شکل :

The screenshot shows a Linux desktop environment with a dark purple background. On the left, there's a file manager window showing several files in the 'Desktop' folder, including 'Q5.sh', 'Q1.sh', 'Q2.sh', 'Q3.sh', 'Q4.sh', 'Q5.sh', 'Q6.sh', 'Q7.sh', 'Q8.sh', 'Q9.sh', 'Q10.sh', and 'Q11.sh'. In the center, there's a terminal window titled 'Q5.sh' with the command 'mahan@mahan-VirtualBox:~/Desktop\$ gedit Q5.sh' and the output 'mahan@mahan-VirtualBox:~/Desktop\$'. To the right, there's a code editor window titled 'Q5.sh' containing the following Bash script:

```
#!/bin/bash
#array=( "10" "20" "30" "40" "50" "60" "70" "80" "90" "100" )
#array=( "10" "20" "30" "40" "50" "60" "70" "80" "90" "100" "110" )
array=( "10" "20" "30" "40" "50" "60" "70" "80" "90" "100" "110" "120" )
flag=1
for (( i=0; i < ${#array[@]}; i++ ))
do
    flag=0
    for (( j=0; j<${#array[@]}-1-i; j++ ))
    do
        if [[ ${array[j]} > ${array[j+1]} ]]; then
            temp=${array[j]}
            array[j]=${array[j+1]}
            array[j+1]=$temp
            flag=1
        fi
    done
done
# printing array
for (( i=0; i<${#array[@]}; i++ ))
do
    echo -ne "${array[i]} "
done
echo -e "\n"
```

حال به جزئیات کد می پردازیم :

```
#!/bin/bash
argArray=( "$@" )
arrayLen="${#argArray[@]}"

flag=1
for (( i=0; i < arrayLen-1 && flag==1 ; i++ ))
do
    flag=0
    for (( j=0; j<arrayLen-i-1; j++ ))
    do
        if (( ${argArray[$j]} > ${argArray[$j+1]} )); then
            temp=${argArray[$j]}
            argArray[$j]=${argArray[$j+1]}
            argArray[$j+1]=$temp
            flag=1
        fi
    done
done

# printing array
for (( l=0; l<arrayLen; l++ ))
do
    echo -ne "${argArray[$l]} "
done
echo -e "
```

حال با استفاده از دستور `chmod +x Q5.sh` به این فایل قابلیت اجرایی نیز می دهیم و با استفاده از دستور `ls -ltrh` چک می کنیم که این کار به درستی انجام شده باشد :

```
Q mahan@mahan-VirtualBox: ~/Desktop
mahan@mahan-VirtualBox: ~/Desktop$ chmod +x Q5.sh
mahan@mahan-VirtualBox: ~/Desktop$ ls -ltrh
total 36K
drwxr-xr-x 10 root root 4.0K سر ام 10 16:08 h
-rwxr-xr-x 1 mahan mahan 548 لیرو 15 18:47 Q1.sh
-rwxr-xr-x 1 mahan mahan 283 لیرو 15 19:28 Q2.sh
-rwxr-xr-x 1 mahan mahan 359 لیرو 15 21:51 Q3.sh
drwxr-xr-x 2 mahan mahan 4.0K لیرو 15 23:46 M
-rwxr-xr-x 1 mahan mahan 182 لیرو 15 23:47 Q4.sh
-rw-r--r-- 1 mahan mahan 71 لیرو 15 23:55 Q4Test.txt
drwxr-xr-x 2 mahan mahan 4.0K لیرو 16 18:56 Q
-rwxr-xr-x 1 mahan mahan 484 لیرو 16 18:57 Q5.sh
mahan@mahan-VirtualBox: ~/Desktop$
```

حال می خواهیم تمام سناریو های پیاده سازی شده در کد بالا را بررسی کنیم :

```
Q mahan@mahan-VirtualBox: ~/Desktop
mahan@mahan-VirtualBox: ~/Desktop$ ./Q5.sh 8 2 1 7 7 12 78 9
1 2 7 7 8 9 12 78
mahan@mahan-VirtualBox: ~/Desktop$ ./Q5.sh 9 2 3 5 7 11 4 3
2 3 3 4 5 7 9 11
mahan@mahan-VirtualBox: ~/Desktop$
```

همانطور که می بینیم اعداد ورودی داده شده به درستی به صورت سعودی sort شده اند.(توجه شود که در اینجا به صورت سعودی sort کرده ایم)
تحلیل نکات مهم کد :

ابدا با استفاده از متغیر `@$` آرگومان های ورودی کاربر را گرفته ایم و در آرایه `argArray` ذخیره کرده ایم(در واقع متغیر `$@` به تمام آرگومان های ورودی اشاره می کند, `$1, $2, .. $3`، سپس با استفاده از `{#argArray[@]}` طول آرایه را گرفته ایم و در `lenArray` ذخیره کرده ایم، سپس با استفاده از حلقه `for`، `bubble sort` را پیاده سازی کرده ایم و در نهایت حاصل را با استفاده از دستور `echo` چاپ کرده ایم.
نکات مربوط به الگوریتم `Bubble sort` :

در این روش مرتب سازی هر عنصر آرایه فقط با عنصر کناری خود مقایسه می شود و اگر عنصر N ام بزرگ تر از عنصر $N+1$ ام بود جای این دو عنصر عوض می شود این کار تا زمانی که تمام آرایه مرتب شود ادامه می یابد .

REAL NUMBERS CALCULATOR

ماشین حسابی برای اعداد حقیقی طراحی کنید.
برای این کار یک سری تغییرات در کد سوال ۲ ایجاد کرده
ایم.

```
#!/bin/bash

operands_regex='^[-+]?[0-9]+([.][0-9]+)?$'

read -p "Enter first operator: " op1
read -p "Enter operand: " operand
read -p "Enter second operator: " op2

# Checking whether arguments are numeric

if [[ ! "$op1" =~ $operands_regex ]]; then
    echo "\$op1\" is not a real value."
    exit 1
fi

if [[ ! "$op2" =~ $operands_regex ]]; then
    echo "\$op2\" is not a real value."
    exit 1
fi

echo -n "Result: "
case $operand in
    +)
        echo "$op1 + $op2" | bc ;;
    -)
        echo "$op1 - $op2" | bc ;;
    (*)
        echo "$op1 * $op2" | bc ;;
    (/)
        echo "$op1 / $op2" | bc ;;
esac
```

برای اینکه چک کنیم که آرگومان های ورودی کاربر درست هستند از رجکس '\$-[0-9]+.[0-9]+[-+]?[0-9]?' برای تشخیص حقیقی بودن ورودی ها استفاده کرده ایم و با استفاده از دستور case ماشین حساب را پیاده کرده ایم و از bc | برای عملیات های اعداد حقیقی استفاده کرده ایم.

حال تمام سناریو های پیاده سازی شده را بررسی می کنیم :

```
#!/bin/bash
operands_regex='[+-]?[0-9]+([.-][0-9]+)*$'
read -p "Enter first operator: " op1
read -p "Enter operand: " operand1
read -p "Enter second operator: " op2
read -p "Enter second operand: " operand2

# Checking whether arguments are numeric
if [[ ! "$op1" =~ $operands_regex ]] || ! [[ "$op1" =~ ^[0-9]+(\.[0-9]+)*$ ]]; then
    echo "$op1 is not a real value."
    exit 1
fi
if [[ ! "$op2" =~ $operands_regex ]] || ! [[ "$op2" =~ ^[0-9]+(\.[0-9]+)*$ ]]; then
    echo "$op2 is not a real value."
    exit 1
fi

echo -n "Result: "
case $operator in
    +)
        echo "$op1 + $op2" | bc ;;
    -)
        echo "$op1 - $op2" | bc ;;
    *)
        echo "$op1 * $op2" | bc ;;
    /)
        echo "$op1 / $op2" | bc ;;
esac
```

```
mahan@mahan-VirtualBox:~$ ./Q2.sh
Enter first operator: 12.5
Enter operand: 3.2
Enter second operator: 1.2
Result: 15.7
mahan@mahan-VirtualBox:~$ ./Q2.sh
Enter first operator: 8.2
Enter operand: 2
Enter second operator: 2
Result: 6.0
mahan@mahan-VirtualBox:~$ ./Q2.sh
Enter first operator: 1.2
Enter operand: 9
Enter second operator: 9
Result: 108.0
mahan@mahan-VirtualBox:~$ ./Q2.sh
Enter first operator: 9.5
Enter operand: 4
Enter second operator: 4
Result: 38.0
mahan@mahan-VirtualBox:~$
```

همانظور که مشاهده می کنیم، ماشین حساب به درستی پیاده سازی شده است.

OTHER QUESTIONS

چگونگی دسترسی به دهمین آرگومان :

برای انجام این کار باید شماره آرگومان را در داخل {} بگذاریم مثلاً داریم :

echo "{\$10}"

تفاوت سوییچ های -p و -sپ :

به کمک سوییچ -p می توانیم یک رشته را پیش از ورودی گرفتن به عنوان راهنما برای کاربر چاپ کنیم، اما سوییچ -s باعث می شود کاراکترهای وارد شده در خروجی نوشته نشوند و برای وارد کردن رمز خوب مورد استفاده قرار می گیرد.

متغیر \$\$:

در واقع این متغیر pid یا (که عددی یونیک برای process است) مربوط به مفسر shell ای است که در حال حاضر script را دارد run می کند.

متغیر \$@ :

برای این متغیر در واقع به تمام آرگومان های ورودی اشاره می کند ... \$1, \$2, \$3,

THE END