

آزمایش 3

40013045

فاطمه راق

9931043

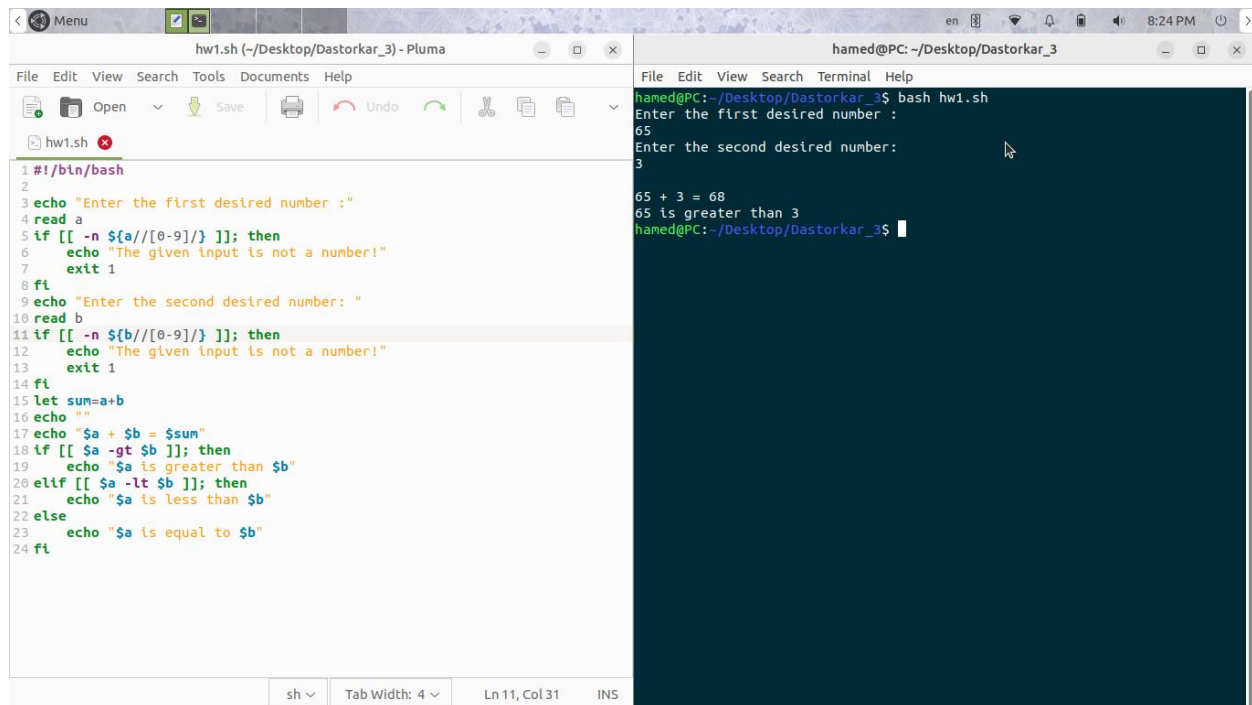
حامد فلاحي

خلاصه

بطور کلی، `bash` مفسر زبان دستورات است. `Shell` یک پردازشگر ماکرو است که دستورات را اجرا میکند. اصطلاح پردازشگر ماکرو به معنای عملکرد است که در آن متن و نمادها برای ایجاد عبارات بزرگتر گسترش میابد. `Unix shell` یک زبان برنامه نویسی و مترجم دستورات است که ویژگی های این زبان برنامه نویسی اجازه میدهد که مجموعه ای غنی از سرویس های `GNU` ترکیب شوند. فایل هایی که حاوی دستورات هستند میتوانند ایجاد شوند و خودشان تبدیل به دستور شوند. حال ما قصد داریم در این آزمایش، باتوجه به نحوه کار با این زبان برنامه نویسی و ویژگی های آن که در دستورکار نیز آمده، تمرین های گفته شده را مورد بررسی قرار داده و آن هارا حل کنیم.

(HW1)

در تمرین 1، از ما خواسته شده که دو عدد بعنوان ورودی به برنامه بدهیم و سپس جمع آن دو عدد را به همراه مقایسه دو عدد به ما بعنوان خروجی بدهد. همچنین در صورت اشتباه، پیغام مناسب برای ما چاپ کند. همانطور که در عکس میبینیم، با استفاده از حالت شرطی که در دستورکار نیز گفته شده، برای `a` و `b` هرکدام `read` مینویسیم و به ازای آنها شرط میگذاریم تا پیغام مناسب به ما بدهد. همچنین از دستور `echo` برای چاپ و ویژگی های دیگر نیز همانند عکس استفاده میکنیم. سپس با استفاده از دستور `let` مقدار جمع اعداد را در متغیر تعریف شده `sum` میریزیم و پس از آن با استفاده از ویژگی های شرطی، مقایسه را بین `a` و `b` انجام میدهیم. همچنین با استفاده از دستور `bash (file)` برنامه را اجرا کرده و نتایج مورد نظر را همانند سمت راست عکس زیر میبینیم.



The image shows a Pluma text editor window on the left and a terminal window on the right. The Pluma window displays a shell script named `hw1.sh` with the following content:

```
1 #!/bin/bash
2
3 echo "Enter the first desired number : "
4 read a
5 if [[ -n ${a//[0-9]/} ]]; then
6     echo "The given input is not a number!"
7     exit 1
8 fi
9 echo "Enter the second desired number: "
10 read b
11 if [[ -n ${b//[0-9]/} ]]; then
12     echo "The given input is not a number!"
13     exit 1
14 fi
15 let sum=a+b
16 echo ""
17 echo "$a + $b = $sum"
18 if [[ $a -gt $b ]]; then
19     echo "$a is greater than $b"
20 elif [[ $a -lt $b ]]; then
21     echo "$a is less than $b"
22 else
23     echo "$a is equal to $b"
24 fi
```

The terminal window on the right shows the execution of the script. The user runs `bash hw1.sh`. The script prompts for the first number, and the user enters `65`. It then prompts for the second number, and the user enters `3`. The script outputs the sum `65 + 3 = 68` and compares the numbers, stating `65 is greater than 3`.

(HW2

در تمرین 2 از ما می‌خواهند که ماشین حساب ساده‌ای با ویژگی `case` بسازیم. در این تمرین نیز از ویژگی‌هایی که در تمرین 1 استفاده شد، استفاده می‌کنیم. می‌توانستیم در ابتدای کد از ویژگی حلقه استفاده نکنیم و دائماً عملیات را در حلقه نگه نداریم... اما این کار را بر فرض انجام دادیم و مشکلی ندارد. سپس ورودی‌های `a` و `b` و ورودی عملگر `op` را می‌خوانیم تا عملیات را با استفاده از آن روی هر دو عدد آرگومان اعمال کنیم. همانطور که می‌بینیم، پس از اعمال حالت‌ها، در حالت ضرب، از حرف اینگلیسی `x` استفاده شده. دلیلش این است که عملگر ضرب `*` در این زبان برنامه‌نویسی در اینجا به معنای هر چیزی غیر از کلیدها می‌باشد (همانند `else`) و ما این کار را برای حل این مشکل انجام دادیم و از آن نتیجه گرفتیم. همانطور که می‌بینید، خروجی این کد نیز به ازای اکثر عملگرها نمایش داده شده است. همچنین حواسمان باشد که خروجی عملیات تقسیم، یک عدد آرگومان و صحیح است و به عدد پایینتر گرد میشود. (مثلاً $64.843 \rightarrow 64$)

```
1 #!/bin/bash
2
3 while(true)
4 do
5 echo ""
6 echo "first number:"
7 read a
8 echo "second number:"
9 read b
10 echo "operator (+ - / x e(Exit)):"
11 read op
12 echo ""
13 case $op in
14 +)
15     let sum=a+b
16     echo "$a + $b = $sum"
17 ;;
18 -)
19     let sum=a-b
20     echo "$a - $b = $sum"
21 ;;
22 /)
23     let sum=a/b
24     echo "$a / $b = $sum"
25 ;;
26 x)
27     let sum=a*b
28     echo "$a * $b = $sum"
29 ;;
30 e)
31     exit 1
32
```

```
hamed@PC: ~/Desktop/Dastorkar_3$ bash hw2.sh
first number:
13
second number:
65
operator (+ - / x e(Exit)):
x
13 * 65 = 845

first number:
453
second number:
34
operator (+ - / x e(Exit)):
x
453 * 34 = 15402

first number:
7674
second number:
756
operator (+ - / x e(Exit)):
/
7674 / 756 = 10

first number:
^C
hamed@PC:~/Desktop/Dastorkar_3$
```

(HW3

در این تمرین به ما گفته شده که برنامه نوشته شده طوری باشد که ورودی یک عدد چند رقمی و خروجی آن عدد صحیح را برای ما معکوس کند و پس از آن، جمع تک تک ارقام چند رقمی را برای ما حساب کند. همانطور که میبینید، ما برای معکوس کردن نیاز به عملگر باقی مانده و تقسیم و همچنین ویژگی حلقه داریم تا به تعداد ارقام این عدد، در حلقه بچرخد و عملیات لازم را انجام دهد. عملیات از این قرار است که ابتدا باقیمانده آن عدد به 10 گرفته شود تا تک تک ارقام را به ازای هر حلقه بدست آوریم. سپس چون خروجی همیشه با تک رقم جدا شده جمع میشود، ارقام کم ارزش باید تبدیل به بیشترین ارزش شوند و به همین دلیل به ازای هر حلقه در 10 ضرب میکنند و با رقم جدید جمع میکنند تا اولویت آن به ازای هر حلقه بالاتر رود. پس از آن نیز عدد ورودی تقسیم بر 10 میشود تا یک واحد از کم ارزش ها کم شود و رقم بعدی جلو آید و پس از آن به ازای هر حلقه، رقم هارا جمع میکند. همانطور که میبینیم، در سمت راست عکس، نتایج مورد نظر میبینیم و میبینیم درست اجرا میشود و خروجی مورد نظر را دریافت میکنیم.

```
hw3.sh (~/.Desktop/Dastorkar_3) - Pluma
1 #!/bin/bash
2
3 while(true)
4 do
5 echo "Enter your number:"
6 read x
7 out=0
8 sum=0
9 while [ $x -gt 0 ]
10 do
11     digit=$(expr $x % 10)
12     out=$(expr $out * 10 + $digit)
13     x=$(expr $x / 10)
14     sum=$(expr $sum + $digit)
15 done
16 echo ""
17 echo "Reverse: $out"
18 echo "Sum of digits: $sum"
19 echo ""
20 done

hamed@PC: ~/Desktop/Dastorkar_3
hamed@PC:~/Desktop/Dastorkar_3$ bash hw3.sh
Enter your number:
853450010

Reverse: 10054358
Sum of digits: 26

Enter your number:
4537

Reverse: 7354
Sum of digits: 19

Enter your number:
^C
hamed@PC:~/Desktop/Dastorkar_3$
```

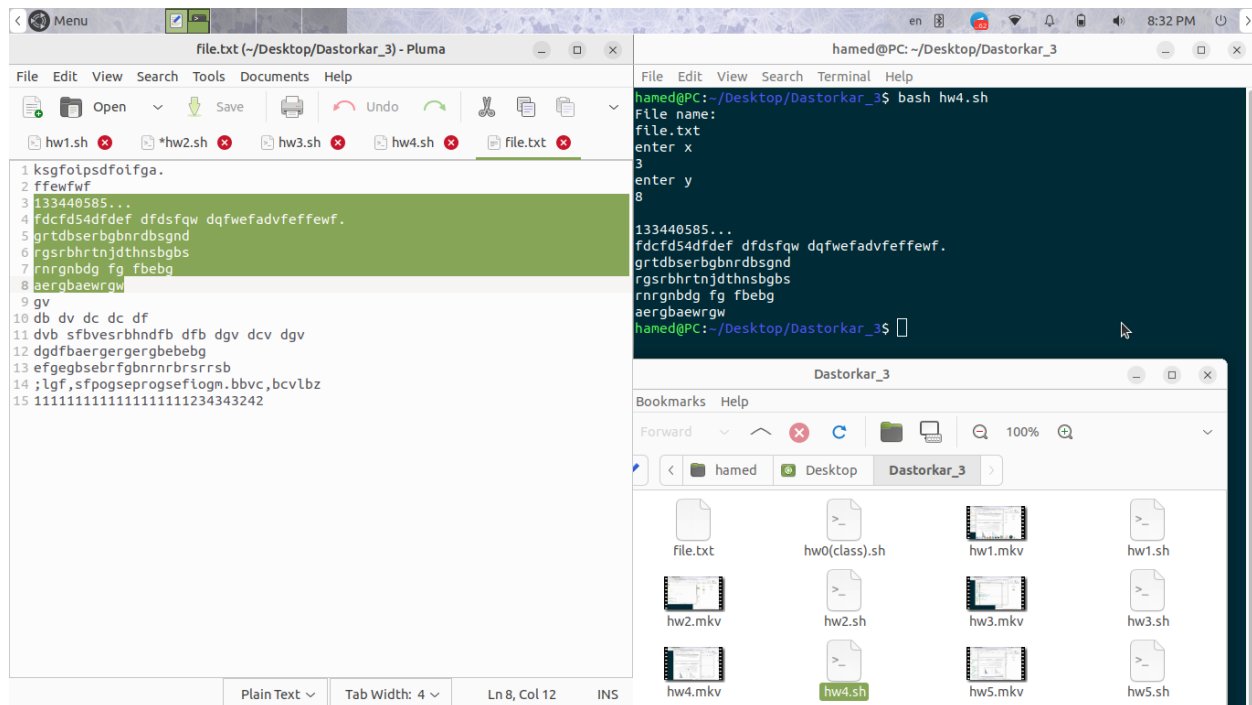
(HW4

در این قسمت نیز از ما میخواهد که یک فایل را بعنوان ورودی بدهیم و سپس از خط x تا خط y که میخواهیم برایمان چاپ شود را ورودی میدهیم. همانطور که میبینیم، در این کد از دستورات `head` و `tail` استفاده شده. وظیفه این دو دستور به ترتیب اینگونه است که باتوجه به اسم فایلی که دریافت میکنند، از خط x به تعداد $(y-x+1)$ خطوط نوشته فایل را به ما نمایش میدهند و در نهایت خروجی را در عکس مشاهده میکنیم که خطوط نمایش داده شده در ترمینال، همانند نوشته موجود در فایل میباشد.

```
hw4.sh - Notepad
File Edit View

#!/bin/bash

echo "File name:"
read file
echo "enter x"
read x
echo "enter y"
read y
echo ""
tail -n +$x $file | head -n (($y-$x+1))
```



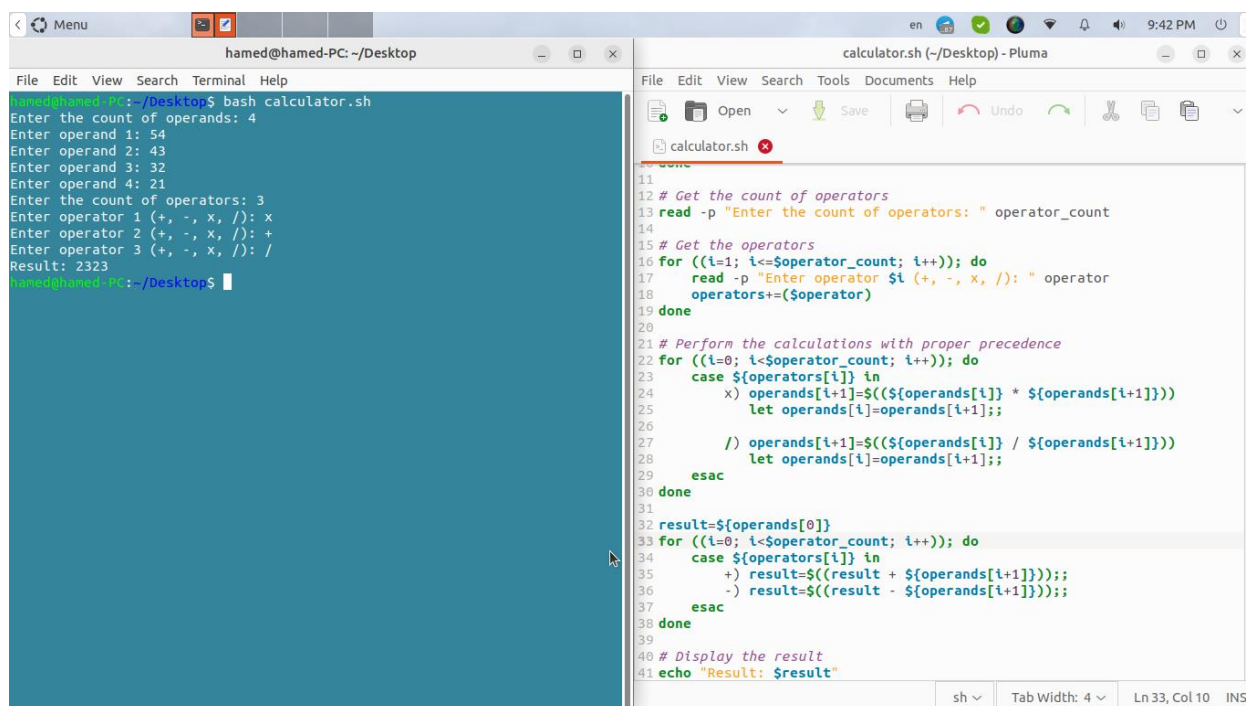
(HW5)

در این تمرین از ما میخواهد که باتوجه به اشکال داده شده به ترتیب از عدد 1 تا 3 که بعنوان ورودی می‌دهیم، این اشکال را برای ما چاپ کند. ابتدا باید از تکنین `case` استفاده کنیم. سپس به ازای عدد 1 و شکل 1، ما باید از 2 حلقه تو در تو استفاده کنیم تا باتوجه به اینکه در هر خط یک عدد اضافه میشود، به ازای `i` در هر خط، کمتر از شماره خط، خود `i` را چاپ کنیم. در شکل 2 و عدد 2، ما باید این لوزی را به 2 مثلث برعکس تقسیم کرده و یک الگوریتم برای یک مثلث ایجاد کنیم. همچنین میبینیم که درون یک حلقه، 2 حلقه وجود دارد که حلقه اولی، تعداد جای خالی و حلقه دومی، تعداد نقطه هارا چاپ میکند. پس از ایجاد یک مثلث، این الگوریتم را به کلی کپی کرده و 2 حلقه درون حلقه اصلی را باهم جابجا میکنیم و با کمی تصحیح جای مثلث، نتیجه صحیح را بدست می آوریم. در شکل 3 و عدد 3، الگوریتم کاملاً شبیه الگوریتم شکل 1 میباشد. با این تفاوت که بجای `i`، علامت `|` پرینت میشود و انتهای حلقه از `_` استفاده میکنیم و نتیجه صحیح را بدست می آوریم. همانطور که در خروجی ترمینال میبینیم، نتایج کاملاً صحیح و درستی به ما بعنوان خروجی میدهد.

The screenshot shows a dual-monitor setup. The left monitor displays a text editor window titled 'hw5.sh (~/.Desktop/Dastorkar_3) - Pluma'. The editor shows a shell script with nested loops and echo statements. The right monitor displays a terminal window titled 'hamed@PC: ~/.Desktop/Dastorkar_3'. The terminal shows the execution of the script, with input prompts and output numbers. The bottom status bar of the terminal window shows 'sh ~', 'Tab Width: 4 ~', 'Ln 19, Col 24', and 'INS'.

(HW6) (امتیازی)

در این تمرین همانطور که در ویدیو گفته شد، باید ماشین حسابی درست کنیم که به ترتیب تعداد اعداد دریافتی را وارد کرده. سپس اعداد را وارد میکنیم. در مرحله بعد تعداد عملگرها را وارد کرده. سپس خود عملگرها را وارد میکنیم (حواسمان باشد که چون عملگرها به ترتیب میان اعداد قرار میگیرند، پس معمولاً تعداد عملگرهای وارد شده برابر با تعداد اعداد منهای 1 میباشد). حالا چون که باید یک ماشین حساب باشد، پس اولویت انجام عملیاتها با ضرب و تقسیم هاست. به همین دلیل اول حلقه برای ضرب و تقسیم گذاشته. بعد از اتمام آن حلقه، حلقه ای دیگر برای جمع و تفریق میگذاریم و نتیجه هارا با `result` جمع میکنیم. نتیجه ران شده به شکل زیر است:



The image shows a terminal window on the left and a text editor window on the right. The terminal window displays the execution of the `calculator.sh` script. The user enters the count of operands (4), followed by four operands (54, 43, 32, 21), and then the count of operators (3). The script prompts for three operators (+, -, x, /). The user enters '+', '-', and '/'. The final result displayed is 2323.

The text editor window shows the source code of `calculator.sh`. The script uses `read` to get the count of operands and operators, and `for` loops to process each operand and operator. It uses `case` statements to handle multiplication, division, addition, and subtraction. The final result is stored in the `result` variable and displayed using `echo`.

```
11
12 # Get the count of operators
13 read -p "Enter the count of operators: " operator_count
14
15 # Get the operators
16 for ((i=1; i<=$operator_count; i++)); do
17     read -p "Enter operator $i (+, -, x, /): " operator
18     operators+=($operator)
19 done
20
21 # Perform the calculations with proper precedence
22 for ((i=0; i<$operator_count; i++)); do
23     case ${operators[i]} in
24         x) operands[i+1]=$(( ${operands[i]} * ${operands[i+1]} ))
25             let operands[i]=operands[i+1];;
26         /) operands[i+1]=$(( ${operands[i]} / ${operands[i+1]} ))
27             let operands[i]=operands[i+1];;
28         esac
29     done
30
31 result=${operands[0]}
32 for ((i=0; i<$operator_count; i++)); do
33     case ${operators[i]} in
34         +) result=$((result + ${operands[i+1]}));;
35         -) result=$((result - ${operands[i+1]}));;
36         esac
37     done
38
39
40 # Display the result
41 echo "Result: $result"
```