

Practical Algorithms and Sums Assignment

1.Pascal Triangle

Step 1: Create a directory names Algos

```
santhosh@ccb228d15332571:~$ mkdir algos
santhosh@ccb228d15332571:~$ touch pascal_triangle.sh
```

Step 2: Editing the saved Pascal_triangle File using nano command.

```
santhosh@ccb228d15332571:~$ nano pascal_triangle.sh
```

Step 3: Providing the Pascal_triangle shell script.

Purpose: The purpose of this shell script is to generate and display a Pascal triangle for a given number of rows.

- `pastri()` defines the Pascal triangle generation function.
- It uses `r=$1` to accept the number of rows as an argument.
- The loop `for((i=0;i<r;i++))` iterates through the number of rows.
- The loop `for((s=1;s<r-i;s++))` ensures proper formatting by adding spaces to center the triangle.
- The loop `for((j=0;j<=i;j++))` calculates the binomial coefficients for each position in the row.
- If the current element is the first element of the row, its value is set to 1.
- For other elements, the binomial coefficient is calculated using:
- The computed value is printed using `echo -n`, which prevents a newline from being added after each value, ensuring they appear on the same row.
- A newline is added at the end of each row using `echo`.
- The script prompts the user to enter the number of rows using `read r`.
- The user-provided input is passed as an argument to the `pastri` function.

```
santhosh@ccb228d15332571: ~
GNU nano 7.2                                pascal_triangle.sh
Shell program to print Pascal triangle

pascal()
{
  r=$1
  c=1
  for ((i=0;i<r;i++))
  do
    for ((s=1;s<r-i;s++))
    do
      echo -n " "
    done
    for ((j=0;j<=i;j++))
    do
      if [ $j -eq 0 -o $i -eq 0 ]
      then
        c=1
      else
        c=$((c*(i-j+1)/j))
      fi
      echo -n $c " "
    done
    echo
  done

  echo "enter the number of rows:"
  read r
  pascal $r
}

Read 31 lines
Execute Justify
Location Go To Line
M-U Undo
M-E Redo
Activate Windows
Go to Settings to activate Windows.
Set Mark
M-A To Bracket
M-B Where Was
Copy
```

Step 4: Provide the execute permission to the file.

```
santhosh@ccb228d15332571:~$ chmod +x pascal_triangle.sh
santhosh@ccb228d15332571:~$ ./pascal_triangle.sh
```

Step 5: Output

```
santhosh@ccb228d15332571:~$ ./pascal_triangle.sh
enter the number of rows:
4
      1
     1 1
    1 2 1
   1 3 3 1
santhosh@ccb228d15332571:~$ ./pascal_triangle.sh
enter the number of rows:
8
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
santhosh@ccb228d15332571:~$ ./pascal_triangle.sh
enter the number of rows:
12
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
santhosh@ccb228d15332571:~$ nano pascal_triangle.sh
santhosh@ccb228d15332571:~$
```

2. Personal Message

Step 1: Touch a file name Personal message and edit it.

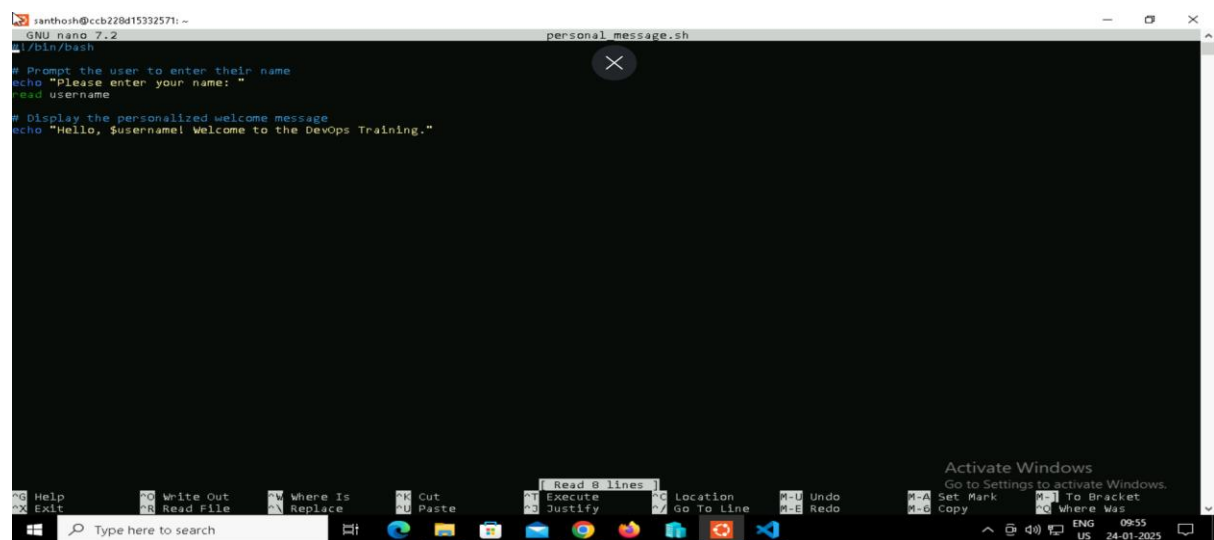
```
santhosh@ccb228d15332571:~$ touch personal_message.sh
santhosh@ccb228d15332571:~$ nano personal_message.sh
```

Step 2: Provide the shell script inside the file.

Purpose: This document explains a simple shell script that interacts with the user by asking for their name and greeting them with a personalized message.

Shebang (#!/bin/bash):

- Specifies that the script should be executed in the bash shell.
- The echo "lease enter your name" command displays a message prompting the user to input their name.
- The read Username command takes the user input and stores it in the variable Username.
- The echo "Hello \$Username! Welcome to the DevOps Training." command uses string interpolation to include the user's input in the greeting message.



Step 3: Run the shell Script.

```
santhosh@ccb228d15332571:~$ ./personal_message.sh
Please enter your name:
Santhosh
```

Step 4: Output

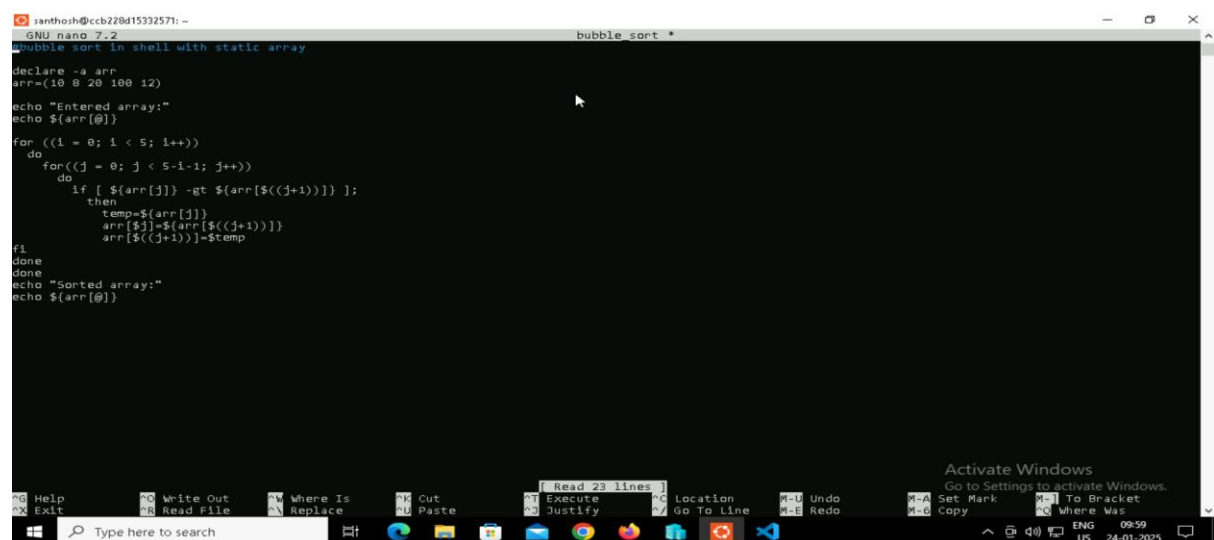
```
santhosh@ccb228d15332571:~$ ./personal_message.sh
Please enter your name:
Santhosh
Hello, Santhosh! Welcome to the DevOps Training.
```

3.Bubble Sort

Step 1: Touch a file name bubble_sort and edit it

```
santhosh@ccb228d15332571:~$ touch bubble_sort
santhosh@ccb228d15332571:~$ nano bubble_sort
```

Step 2: Provide the shell script inside the file.



The screenshot shows a terminal window with the nano text editor open. The file being edited is named 'bubble_sort'. The script content is as follows:

```
GNU nano 7.2 bubble_sort
bubble sort in shell with static array
declare -a arr
arr=(10 8 20 100 12)
echo "Entered array:"
echo ${arr[@]}
for ((i = 0; i < 5; i++))
do
  for ((j = 0; j < 5-i-1; j++))
  do
    if [ ${arr[j]} -gt ${arr[j+1]} ]; then
      temp=${arr[j]}
      arr[j]=${arr[j+1]}
      arr[j+1]=$temp
    fi
  done
done
echo "Sorted array:"
echo ${arr[@]}
```

The terminal window also shows the Windows taskbar at the bottom with various application icons and system tray information.

Step 3: Run the shell Script after providing permissions to execute using chmod.

```
santhosh@ccb228d15332571:~$ chmod +x bubble_sort
santhosh@ccb228d15332571:~$ ./bubble.sort
```

Step 4: Output

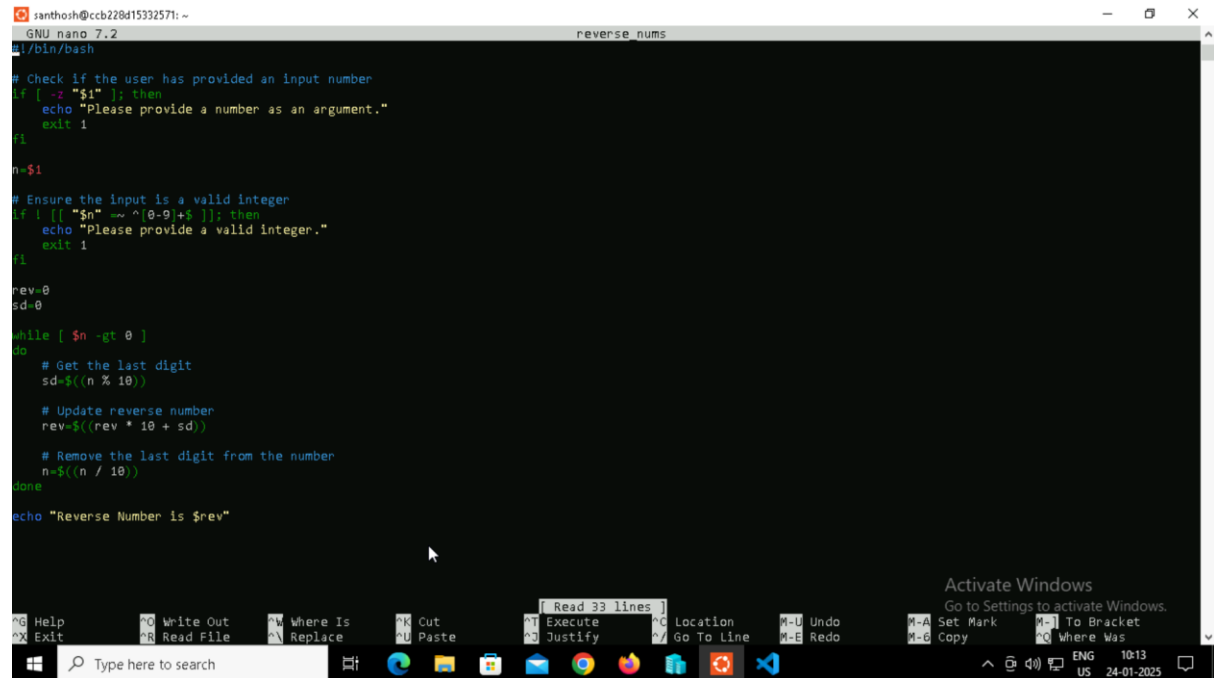
```
santhosh@ccb228d15332571:~$ ./bubble_sort
Entered array:
10 8 20 100 12
Sorted array:
8 10 12 20 100
```

4.Reverse_nums

Step 1: Touch a file name Reverse_nums and edit it

```
santhosh@ccb228d15332571:~$ touch reverse_nums
santhosh@ccb228d15332571:~$ nano reverse_nums
```

Step 2: Provide the shell script inside the file.



```
santhosh@ccb228d15332571: ~
GNU nano 7.2 reverse_nums
# Check if the user has provided an input number
if [ -z "$1" ]; then
    echo "Please provide a number as an argument."
    exit 1
fi

n=$1

# Ensure the input is a valid integer
if ! [[ "$n" =~ ^[0-9]+$ ]]; then
    echo "Please provide a valid integer."
    exit 1
fi

rev=0
sd=0

while [ $n -gt 0 ]
do
    # Get the last digit
    sd=$((n % 10))

    # Update reverse number
    rev=$((rev * 10 + sd))

    # Remove the last digit from the number
    n=$((n / 10))
done

echo "Reverse Number is $rev"
```

Step 3: Run the shell Script after providing permissions to execute using chmod.

```
santhosh@ccb228d15332571:~$ chmod +x reverse_nums
```

Step 4: Output

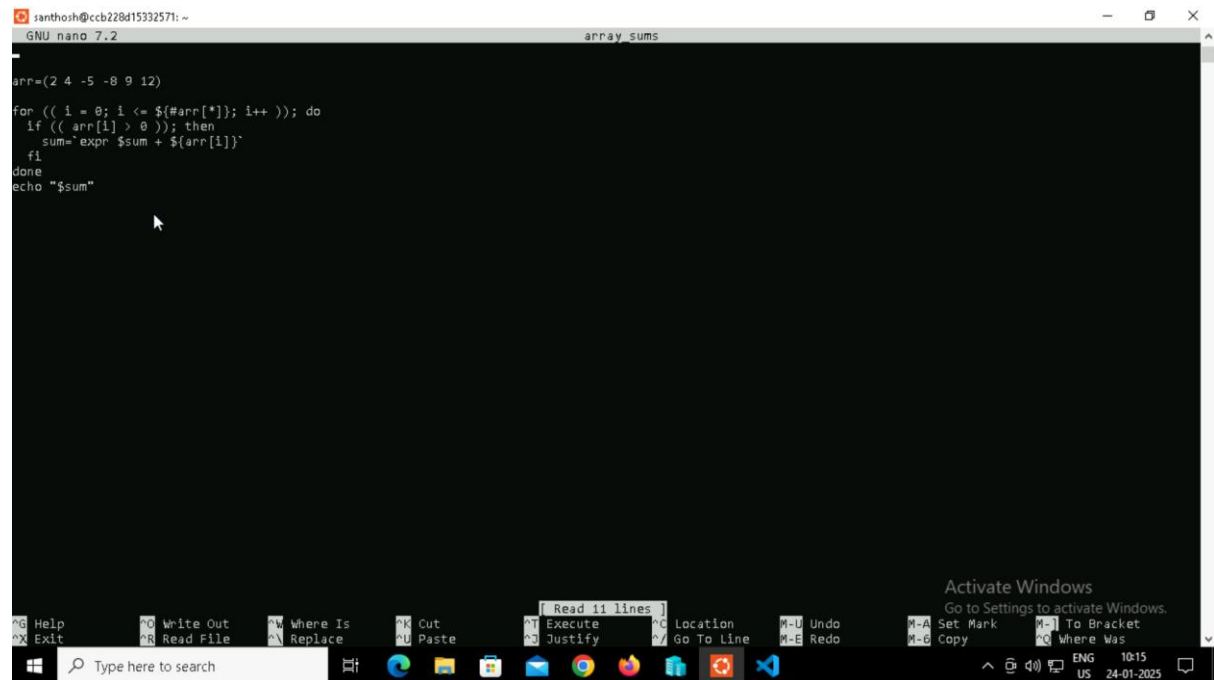
```
12: Command not found
santhosh@ccb228d15332571:~$ ./reverse_nums 98765
Reverse Number is 56789
```

5.Array sums

Step 1: Touch a file name Array_sums and edit it

```
santhosh@ccb228d15332571:~$ touch array_sums
santhosh@ccb228d15332571:~$ nano array_sums
```

Step 2: Provide the shell script inside the file.



```
santhosh@ccb228d15332571: ~
GNU nano 7.2 array_sums

arr=(2 4 -5 -8 9 12)
for (( i = 0; i <= ${#arr[*]}; i++ )); do
    if (( arr[i] > 0 )); then
        sum=$((sum + arr[i]))
    fi
done
echo "$sum"
```

Step 3: Run the shell Script after providing permissions to execute using chmod.

```
santhosh@ccb228d15332571:~$ chmod +x array_sums
```

Step 4: Output

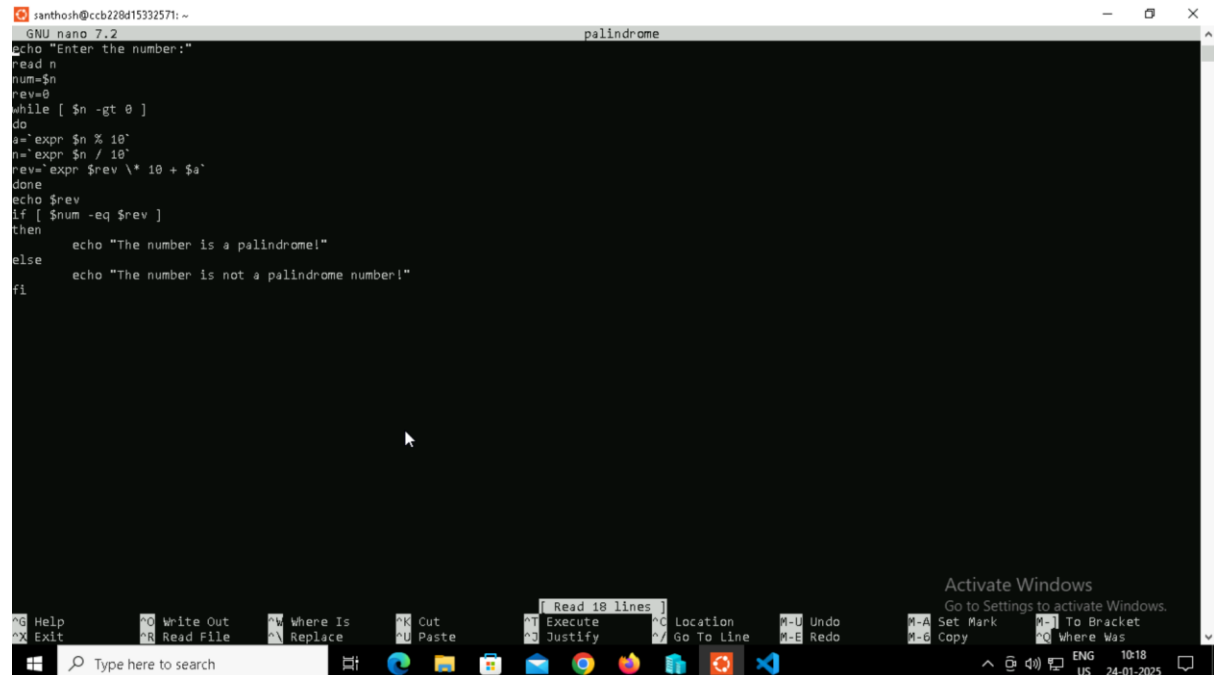
```
santhosh@ccb228d15332571:~$ ./array_sums
27
```

6.Palindrome

Step 1: Touch a file name Palindrome and edit it

```
santhosh@ccb228d15332571:~$ touch palindrome
santhosh@ccb228d15332571:~$ nano palindrome
```

Step 2: Provide the shell script inside the file.



```
GNU nano 7.2 palindrome
echo "Enter the number:"
read n
num=$n
rev=0
while [ $n -gt 0 ]
do
a=`expr $n % 10`
n=`expr $n / 10`
rev=`expr $rev \* 10 + $a`
done
echo $rev
if [ $num -eq $rev ]
then
echo "The number is a palindrome!"
else
echo "The number is not a palindrome number!"
fi
```

Step 3: Run the shell Script after providing permissions to execute using chmod.

```
santhosh@ccb228d15332571:~$ chmod +x palindrome
```

Step 4: Output

```
santhosh@ccb228d15332571:~$ ./palindrome
Enter the number:
12345
54321
The number is not a palindrome number!
```

7. Arithmetic

Shell Script: Integer Addition with Input Validation

Step 1: Create a directory first using mkdir sum and change the directory to the sum directory.

```
santhosh@ccb228d15332571: ~/Demo/sum
santhosh@ccb228d15332571:~/Demo$ mkdir sum
santhosh@ccb228d15332571:~/Demo$ cd sum
```

Step 2: Touch and nano a file named arithmetic.sh

```
santhosh@ccb228d15332571:~/Demo/sum$ touch arithmetic.sh
santhosh@ccb228d15332571:~/Demo/sum$ nano arithmetic.sh
```

Step 3: Code Purpose

1. Prompts the user: Asks the user to enter two integer values.
2. Input Validation: Checks if the user provided input for both values. If either input is empty, an error message is displayed, and the script exits.
3. Calculates Sum: Calculates the sum of the two input integers using the bc command for arbitrary precision arithmetic. Calculates the sum using the expr command for basic integer arithmetic.
4. Displays Results: Prints the calculated sum using both bc and expr.

```
santhosh@ccb228d15332571:~/Demo/sum$ cat arithmetic.sh
#Shell program to add two integer values
#and check if any input is given or not

#!/usr/bin/bash
read -p "Input1 : " inp1
if [[ -z $inp1 ]]
then
    echo "Input 1 cannot be empty, please enter an integer."
    exit
fi

read -p "Input2 : " inp2
if [[ -z $inp2 ]]
then
    echo "Input 2 cannot be empty, please enter an integer."
    exit
fi

bc_val=$(echo "$inp1+$inp2" | bc)
echo "BC Value : $bc_val"

expr_val=$(expr $inp1 + $inp2)
echo "EXPR Value : $expr_val"

santhosh@ccb228d15332571:~/Demo/sum$
```

Step 4: Code Explanation • `#!/usr/bin/bash` specifies that the script should be executed with the Bash interpreter. • `read -p "Input1 : " inp1`: Prompts the user

to enter the first integer and stores it in the inp1 variable. • `read -p "Input2 : "`
`inp2`: Prompts the user to enter the second integer and stores it in the `inp2`
variable. • `if [[-z $inp1]]`: Checks if `inp1` is empty. If empty, displays an error
message and exits with an error code (1). • `if [[-z $inp2]]`: Checks if `inp2` is
empty. If empty, displays an error message and exits with an error code (1). •
`bc_val=$(echo "$inp1 + $inp2" | bc)`: Uses the `bc` command to calculate the sum
with arbitrary precision and stores the result in `bc_val`. • `expr_val=$(expr $inp1
+ $inp2)`: Uses the `expr` command to calculate the sum with basic integer
arithmetic and stores the result in `expr_val`. • `echo "BC Value : $bc_val"`:
Displays the sum calculated using `bc`. • `echo "EXPR Value : $expr_val"`:
Displays the sum calculated using `expr`.

Step 5: Running the file using `./arithmetic.sh`

```
santhosh@ccb228d15332571:~/Demo/sum$ ./arithmetic.sh  
-bash: ./arithmetic.sh: Permission denied
```

Step 6: Providing the execut permissions to the arithmetic shell file.

```
santhosh@ccb228d15332571:~/Demo/sum$ chmod +x arithmetic.sh
```

Step 7: Output of the shell script.

```
santhosh@ccb228d15332571:~/Demo/sum$ ./arithmetic.sh  
Input1 : 12  
Input2 : 14  
BC Value : 26  
EXPR Value : 26
```