NOTRE DAME UNIVERSITY BANGLADESH

CSE-3206 Lab Report-02

Course Title: Operating System Lab

Course Code: CSE-3206

Submitted by:

Name: Istiak Alam

ID: 0692230005101005

Batch: CSE-20

Submission Date: 06-02-25

Lab Task Topic : Familiarization Shell Programming.

Submitted to:

Khorshed Alam
Lecturer, NDUB

❑ Shell Programming :

2.1 Variables :

Code :

```
#!/bin/bash
echo "1. String Variable : (suppose -city- is a variable)"
printf "In Shell Code:-\n\n"
echo "city=Dhaka"
echo 'echo "I live in $city"'
echo " "
echo "Output:-"
city="Dhaka"
echo "I live in $city"
#SAME AS-
echo "${city} is the capital of Bangladesh."
```

Output :

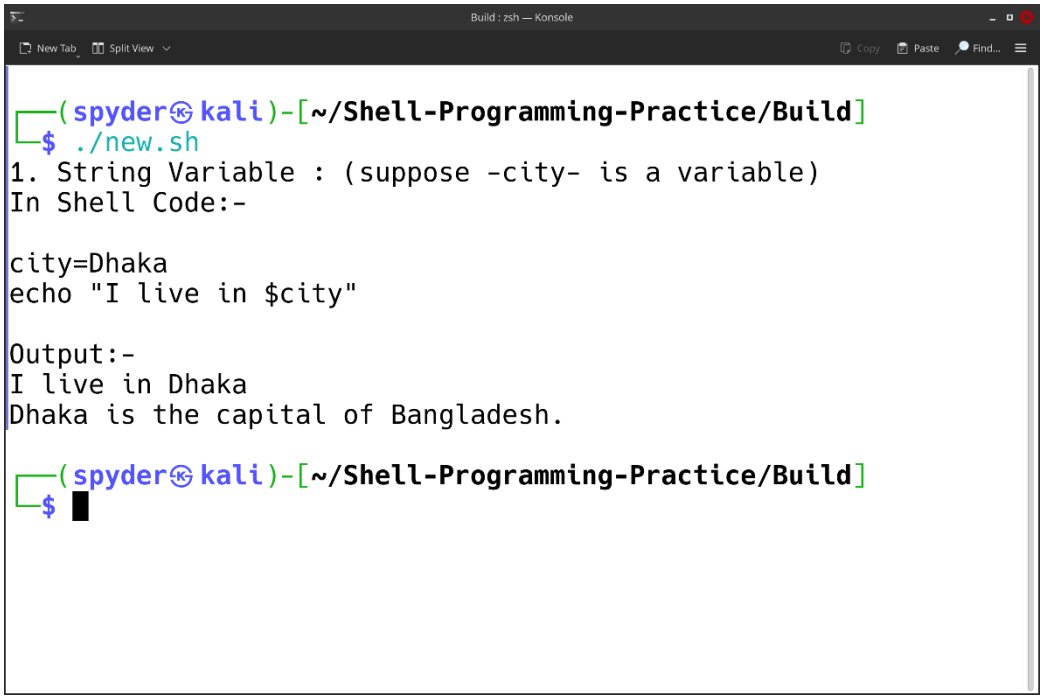

Fig 2.1 : Variable

2.2 User input:

Code :

```
#!/bin/bash
echo "User Input in shell code :"
echo 'read -p "Enter Your Name : " name'
echo 'echo "Hello $name, nice to meet you!"'
echo ""
echo "Output:-"
read -p "Enter Your Name : " name
echo "Hello $name, nice to meet you!"
```

Output :

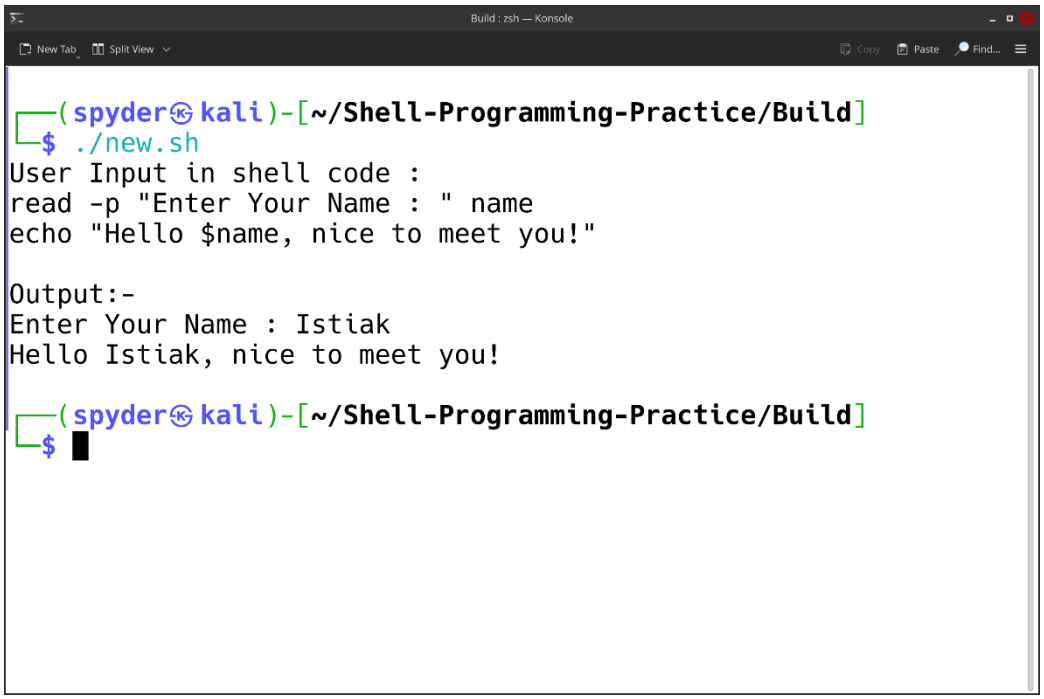

Fig 2.2 : User Input

2.3 IF Statement:

Code :

```
#!/bin/bash
printf "\t\t\tIf Statement :\n"
echo "if name is Brad thn the if will execute."
printf "In Shell Code:\n\n"
echo 'read -p "Enter your name : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad"
fi'
echo ""
echo "Output:-"
read -p "Enter your name : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad."
fi
```

Output :

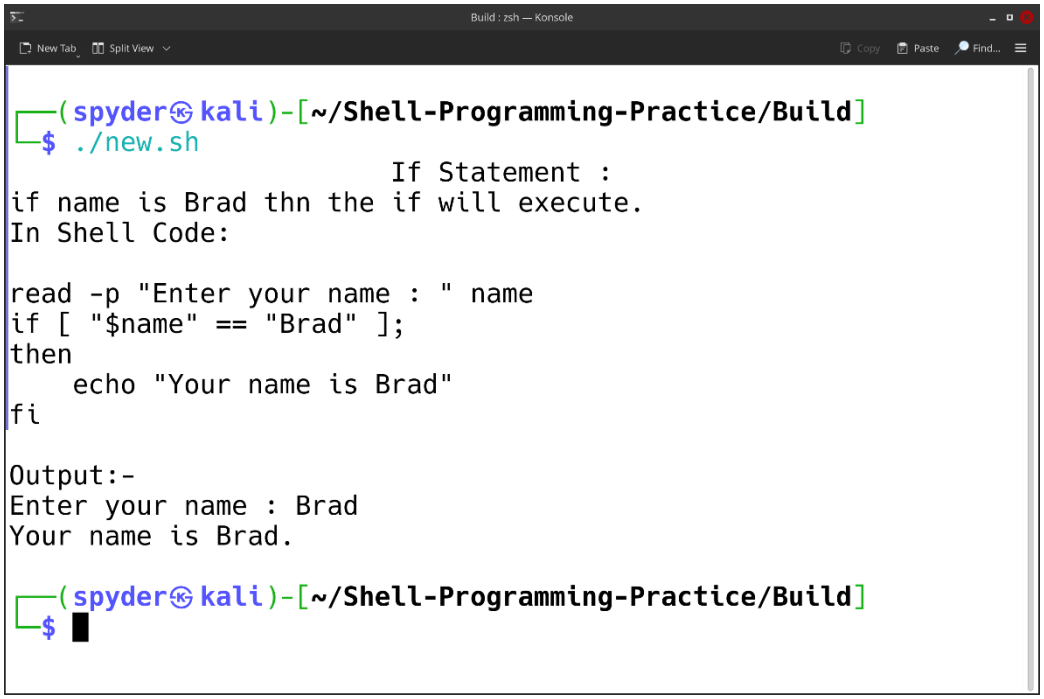

Fig 2.3 : If Statement

2.4 IF-ELSE Statement:

Code :

```
#!/bin/bash
printf "\t\t\t4. If-Else Statement :\n"
printf "In Shell Code:\n\n"
echo 'read -p "Enter a name, Brad or anything : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad"
else
    echo "Your name is NOT Brad"
fi'
printf "\nOutput:-\n"
read -p "Enter a name, Brad or anything : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad"
else
    echo "Your name is NOT Brad"
fi
```

Output :

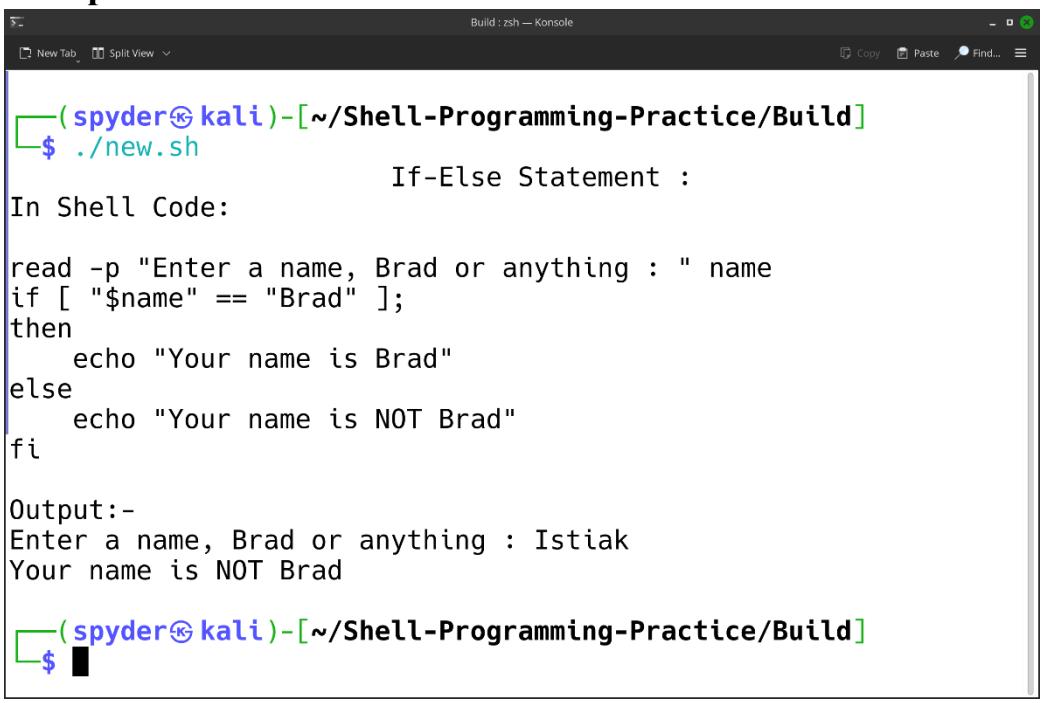

Fig 2.4 : If-else Statement

2.5 ELSE-IF Statement:

Code :

```
#!/bin/bash
printf "\t\t\tElse-If Statement :\n"
printf "In Shell Code:\n\n"
echo 'read -p "Enter a name, Brad / Jack or anything : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad."
elif [ "$name" == "Jack" ];
then
    echo "Your name is Jack."
else
    echo "Your name is nothing."
fi'
printf "\nOutput:-\n"
read -p "Enter a name, Brad / Jack or anything : " name
if [ "$name" == "Brad" ];
then
    echo "Your name is Brad."
elif [ "$name" == "Jack" ];
then
    echo "Your name is Jack."
else
    echo "Your name is nothing."
fi
```

Output :

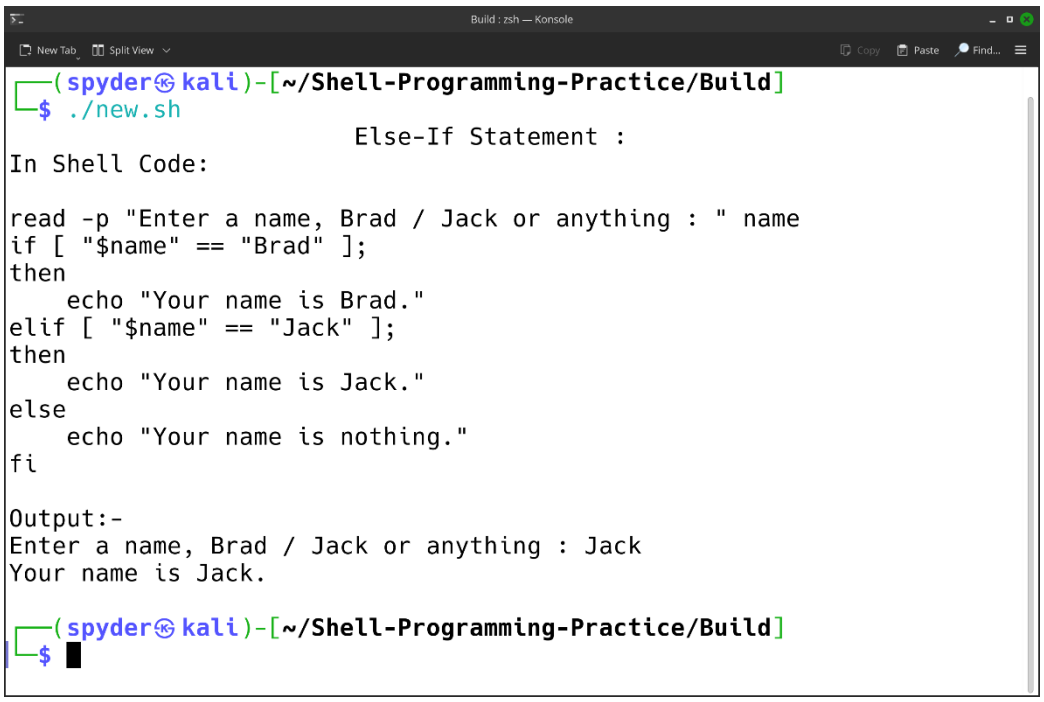

Fig 2.5 : Else-If Statement

2.6 Comparison using IF-ELSE Statement:

Code :

```
#!/bin/bash
printf "\t\t\tComparison :\n"
printf "In Shell Code:\n\n"
echo 'read -p "Enter 1st Number : " num1
read -p "Enter 2nd Number : " num2
if [ $num1 -gt $num2 ];
then
    echo "$num1 is greater than $num2"
else
    echo "$num1 is less than $num2"
fi'
echo ""
echo "Output:-"
read -p "Enter 1st Number : " num1
read -p "Enter 2nd Number : " num2
if [ $num1 -gt $num2 ];
then
    echo "$num1 is greater than $num2"
else
    echo "$num1 is less than $num2"
fi
```

Output :

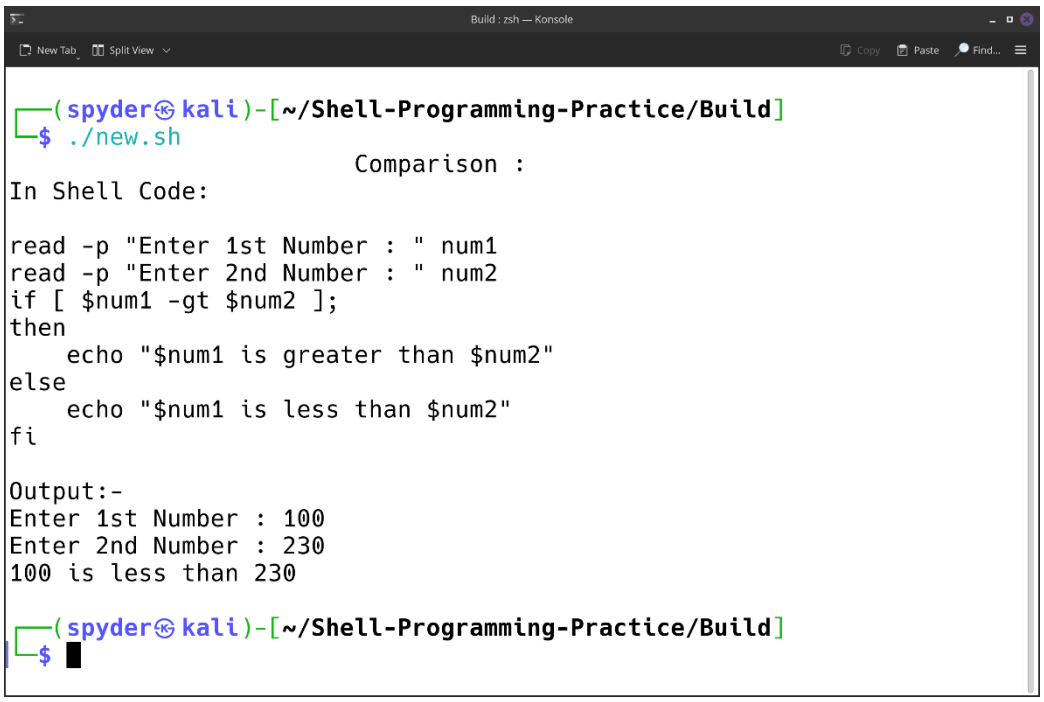

Fig 2.6 : Comparison using If Else

2.7 File Condition :

Code :

```
#!/bin/bash
printf "\t\t\tFile Condition :\n"
printf "Checking if the file exists or not.\n\n"
printf "In Shell Code:\n\n"
echo 'FILE="test.txt"
if [ -e "$FILE" ];
then
    echo "$FILE Exist"
else
    echo "$FILE does not Exist"
    echo "You have to Create $FILE Manually"
fi'
echo ""
echo "Output:-"
FILE="test.txt"
if [ -e "$FILE" ];
then
    echo "$FILE Exist"
else
    echo "$FILE does not Exist"
    echo "You have to Create $FILE Manually"
fi
```

Output :

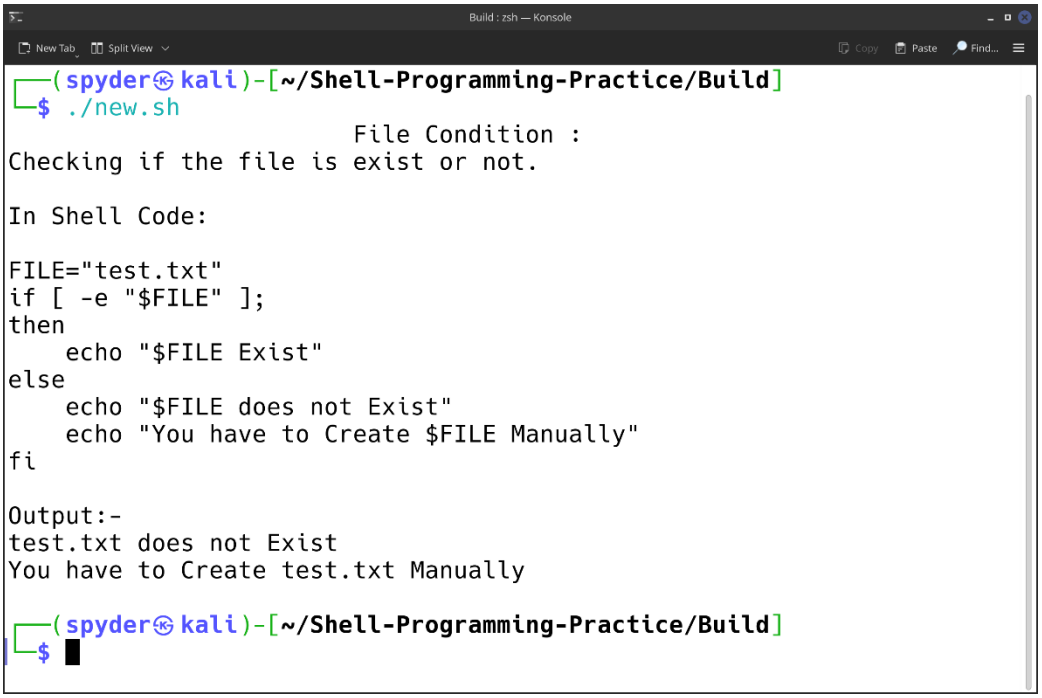

Fig 2.7 : File Condition

2.8 Case Statement:

Code :

```
#!/bin/bash
printf "\t\t\tCase Statement :\n"
printf "In Shell Code:-\n\n"
echo 'read -p "Are you 21 or over? y/n : " ans
case "$ans" in [yY])
echo "You can have Beer  " ;;
[nN])
echo "Sorry, no Drinking  " ;;
*)
echo "Please enter 'y' for YES & 'n' for NO" ;;
esac'
echo ""
echo "Output:-"
read -p "Are you 21 or over? y/n : " ans
case "$ans" in [yY])
echo "You can have Beer :) "
;;
[nN])
echo "Sorry, no Drinking :( "
;;
*)
echo "Please enter 'y' for YES & 'n' for NO"
;;
esac
```

Output :

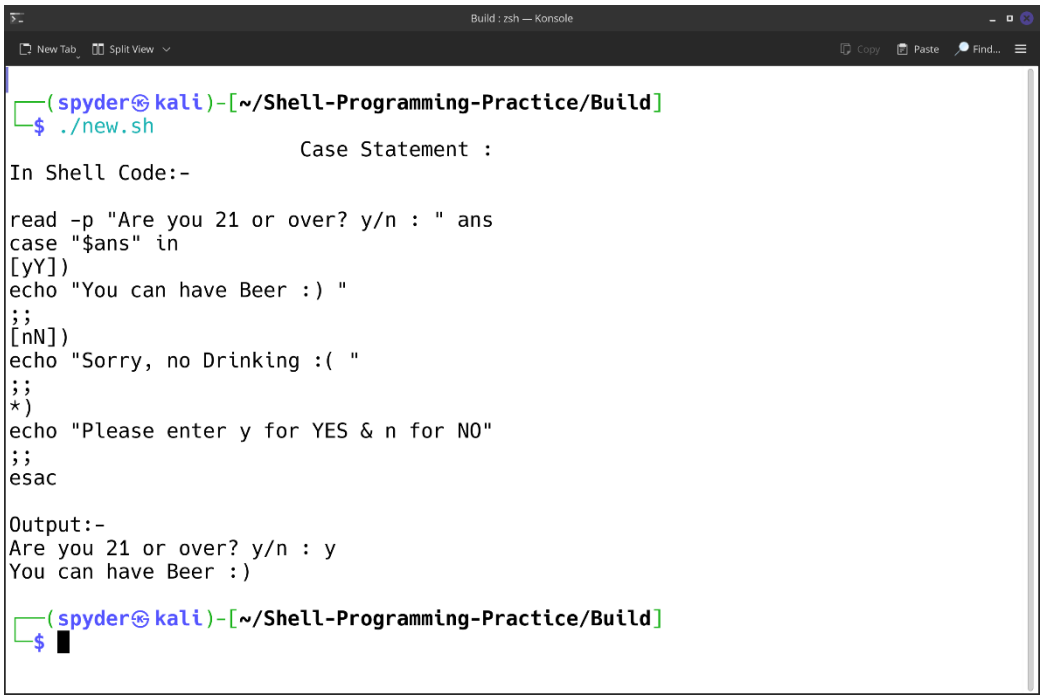

Fig 2.8 : Case Statement

2.9 Pyramid using For Loop Statement:

Code :

```
#!/bin/bash
printf "Pyramid using For-Loop :\n"
read -p "Enter the number of rows: " rows
# Loop to print the pyramid
for (( i=1; i<=rows; i++ ))
do
    # Print spaces before stars
    for (( j=i; j<rows; j++ ))
    do
        echo -n " "
    done
    # Print stars
    for (( j=1; j<=(2*i-1); j++ ))
    do
        echo -n "*"
    done
    # Move to the next line
    echo
done
```

Output :

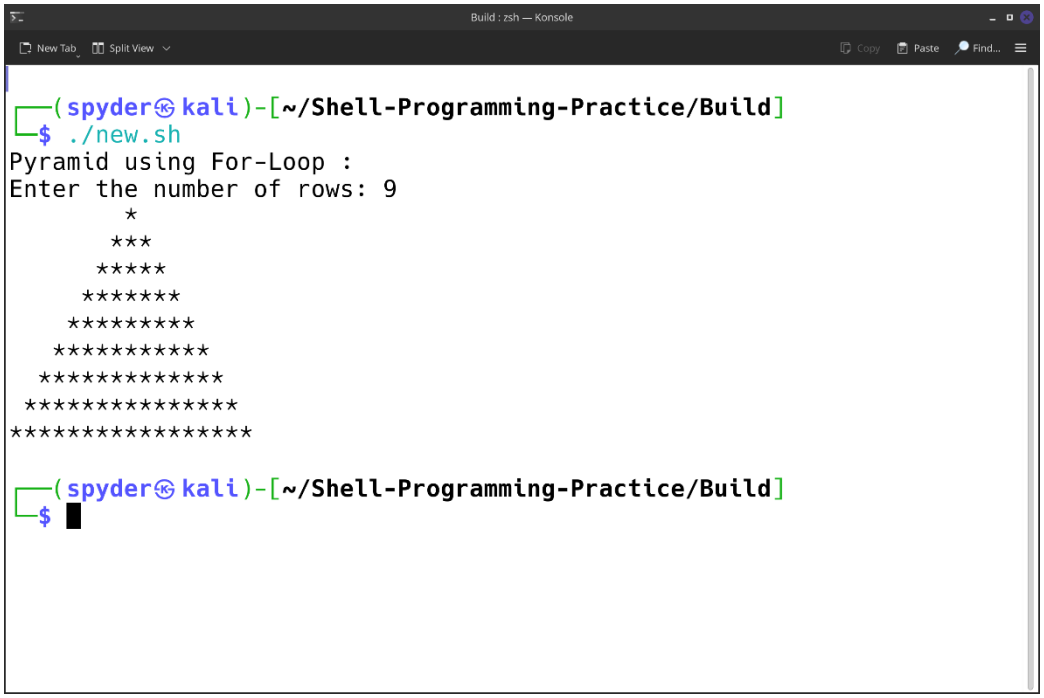

Fig 2.9 : Pyramid using For Loop

2.10 For Loop File Renaming :

Code :

```
#!/bin/bash
printf "\t\t\tFor-Loop RENAME FILES :\n"
printf "\nIn Shell Code:\n\n"
echo 'FILES=$(ls *.txt)
NEW="NewFile"
for FILE in $FILES
do
echo "Renaming $FILE to new-$FILE"
mv $FILE $NEW-$FILE
done'
printf "\nOutput:-\n"
FILES=$(ls *.txt)
NEW="NewFile"
if [ -e "$FILES" ];
then {
for FILE in $FILES
do
echo "Renaming $FILE to new-$FILE"
mv $FILE $NEW-$FILE
done
}
else
    echo "File $FILE does not Exist"
    echo "You have to Create this $FILE Manually"
fi
```

Output :

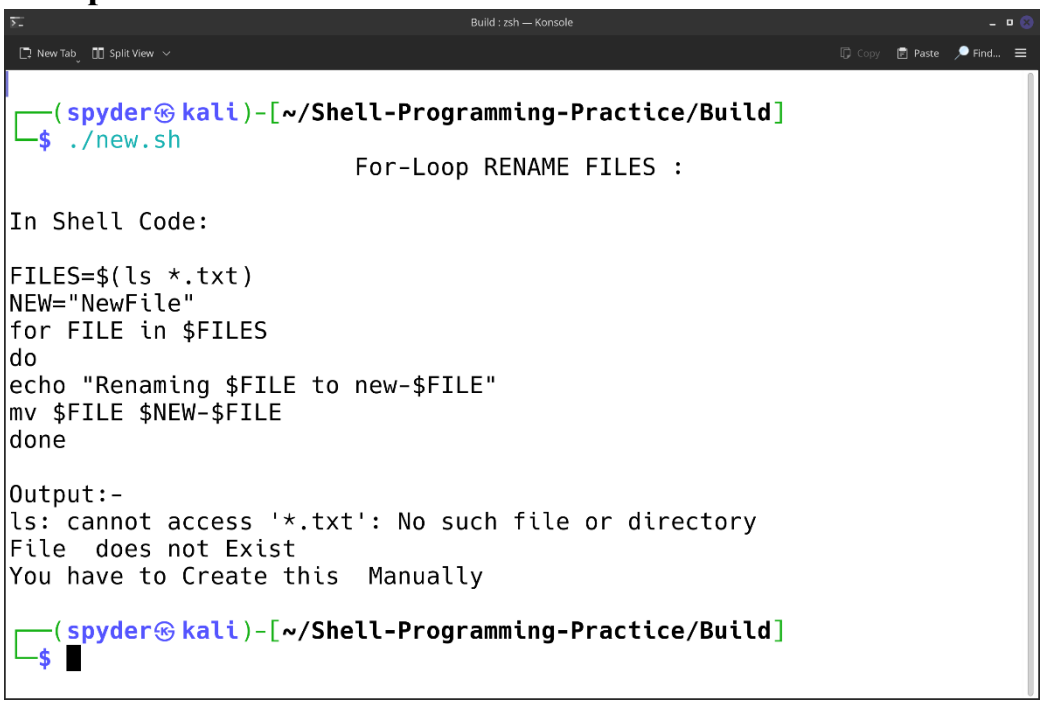

Fig 2.10 : File Rename using For loop

2.11 Until Loop Statement:

Code :

```
#!/bin/bash
printf "\t\t\tUntil Loop Statement :\n"
printf "\nIn Shell Code:\n\n"
echo 'count=5
until [ $count -le 0 ]
do
   echo "Countdown: $count"
   ((count--))
done'
printf "\nOutput:-\n"
count=5
until [ $count -le 0 ]
do
   echo "Countdown: $count"
   ((count--))
done
```

Output :

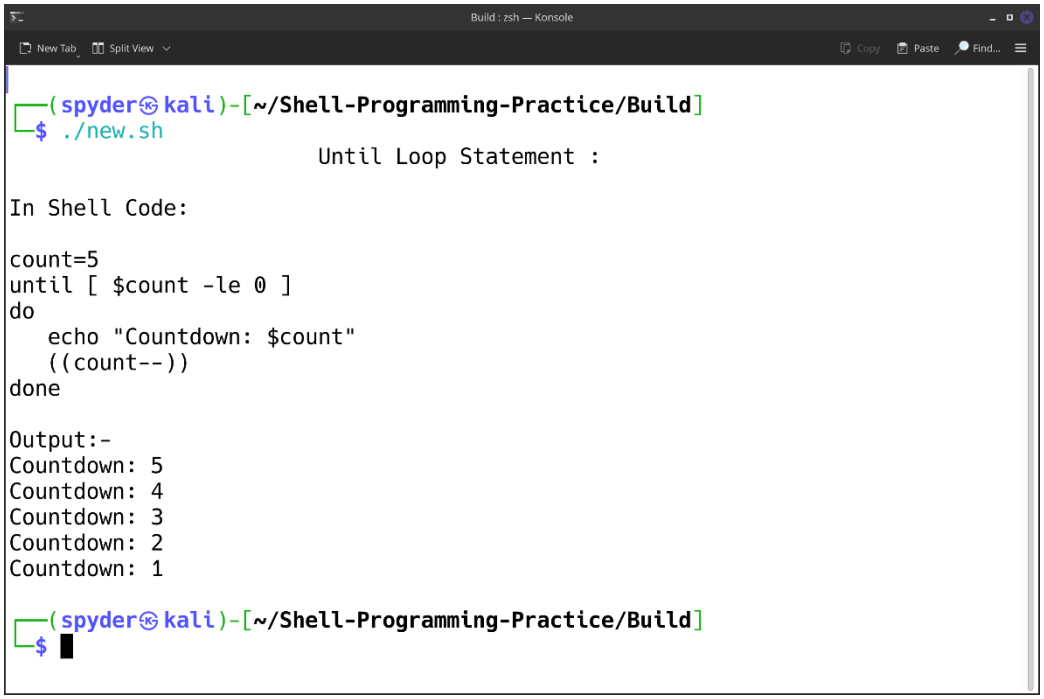

Fig 2.11 : Until Loop Statement

2.12 Function Structure :

Code :

```
#!/bin/bash
printf "\t\t\tFunction Structure :\n"
printf "\nIn Shell Code:\n\n"
#FUNCTION STRUCTURE
echo 'function sayHello( ){
echo "Function Executed"
}
sayHello'
printf "\nOutput:-\n"
function sayHello( ){
echo "Function Executed"
}
sayHello
```

Output :

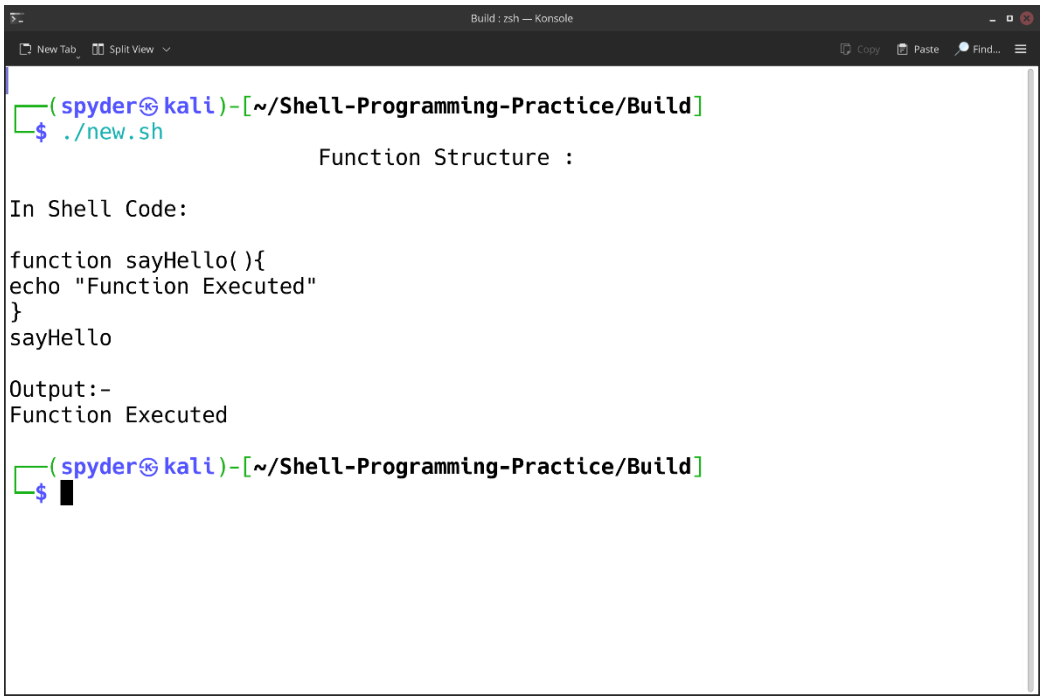

Fig 2.12 : Function Structure

2.13 Function with parameter:

Code :

```
#!/bin/bash
printf "\t\t\tFunction with Parameter :\n"
printf "\nIn Shell Code:\n\n"
echo 'function greet( ){
echo "Hello, I am $1 and I am $2"
}
greet "Brad" "36"'
printf "\nOutput:-\n"
#FUNCTION WITH PARAMETERS
function greet( ){
echo "Hello, I am $1 and I am $2"
}
greet "Brad" "36"
```

Output :

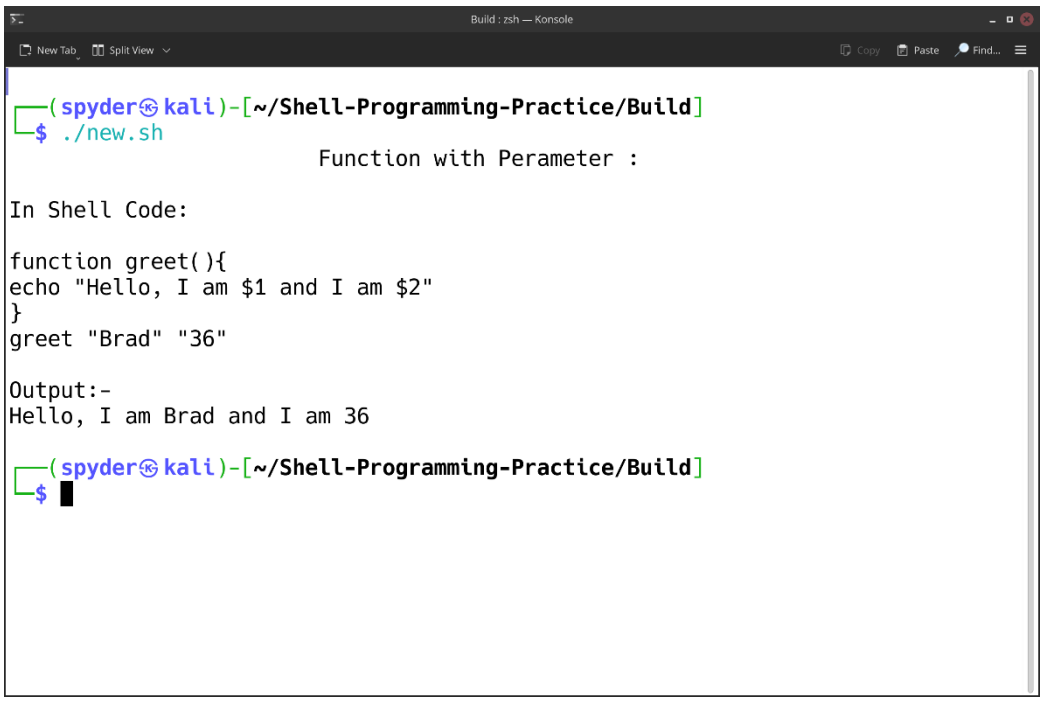

Fig 2.13 : Function with Perameter

2.14 Creating Folder and writing to a file:

Code :

```
#!/bin/bash
printf "\t\t\tCreating Folder and writting to a File :"
printf "\nIn Shell Code:\n\n"
echo 'mkdir Hello
touch "Hello/world.txt"
echo "Hello My beautiful world" >> "Hello/world.txt"
echo "Created Hello/world.txt"'

printf "\nOutput:-\n"
#CREATE FOLDER AND WRITE TO A FILE
mkdir Hello
touch "Hello/world.txt"
echo "Hello My beautiful world" >> "Hello/world.txt"
echo "Created Hello/world.txt"
```

Output :

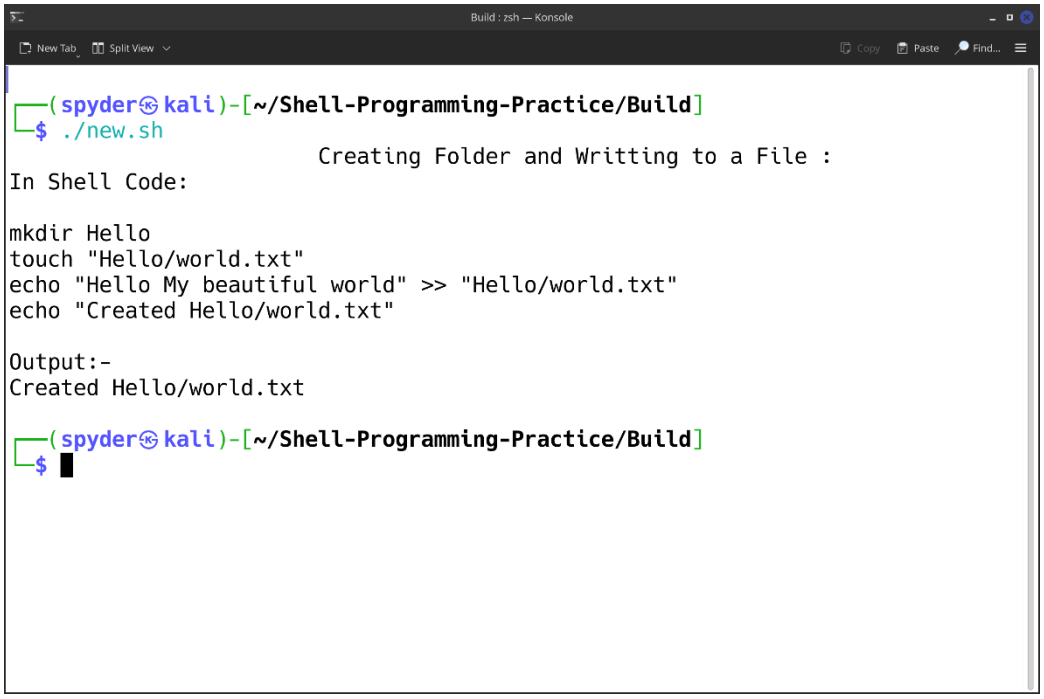

Fig 2.14 : Folder & write in file

2.15 Array Declaration:

Code :

```
#!/bin/bash
printf "\t\t\tArray Declaration\n"
printf "In Shell Code:-\n\n"
echo 'fruits=("apple" "banana" "cherry")'
echo 'echo "Array of fruits : ${fruits[]}"'
printf "\nOutput:-\n"
fruits=("apple" "banana" "cherry")
echo "Array of fruits : ${fruits[]}"
echo ""
```

Output :

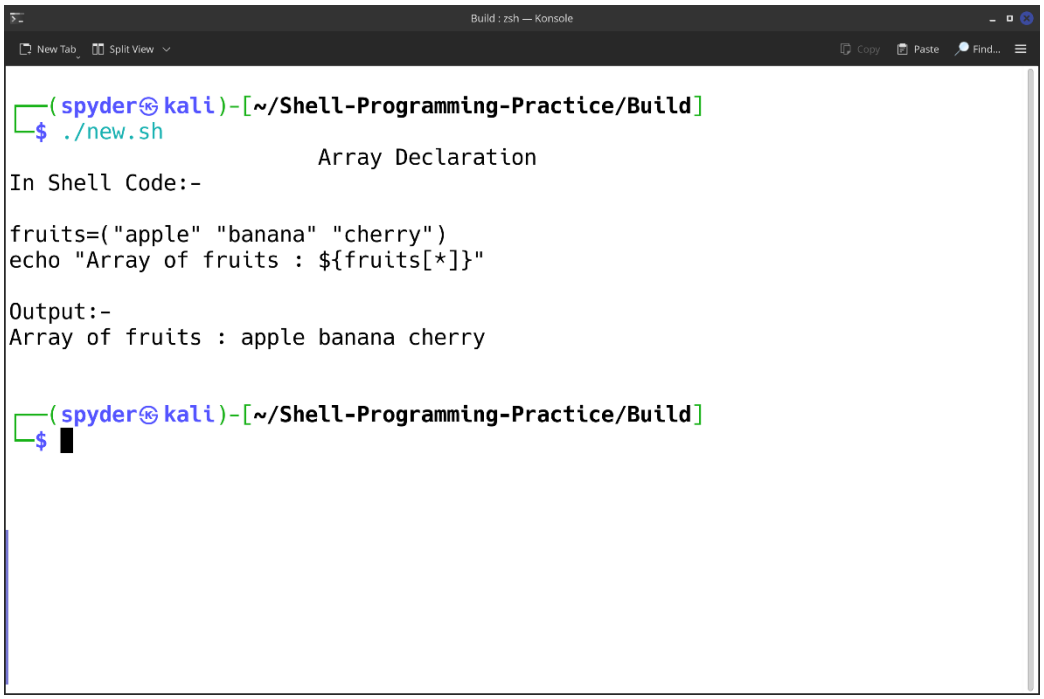

Fig 2.15 : Array Declaration

2.16 Adding elements in Array :

Code :

```
#!/bin/bash
printf "\t\t\tAdding Elements\n"
printf "In Shell Code:-\n\n"
echo 'fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"
echo "Array of fruits : ${fruits[*]}"'
printf "\nOutput:-\n"
fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"
echo "Array of fruits : ${fruits[*]}"
echo ""
```

Output :

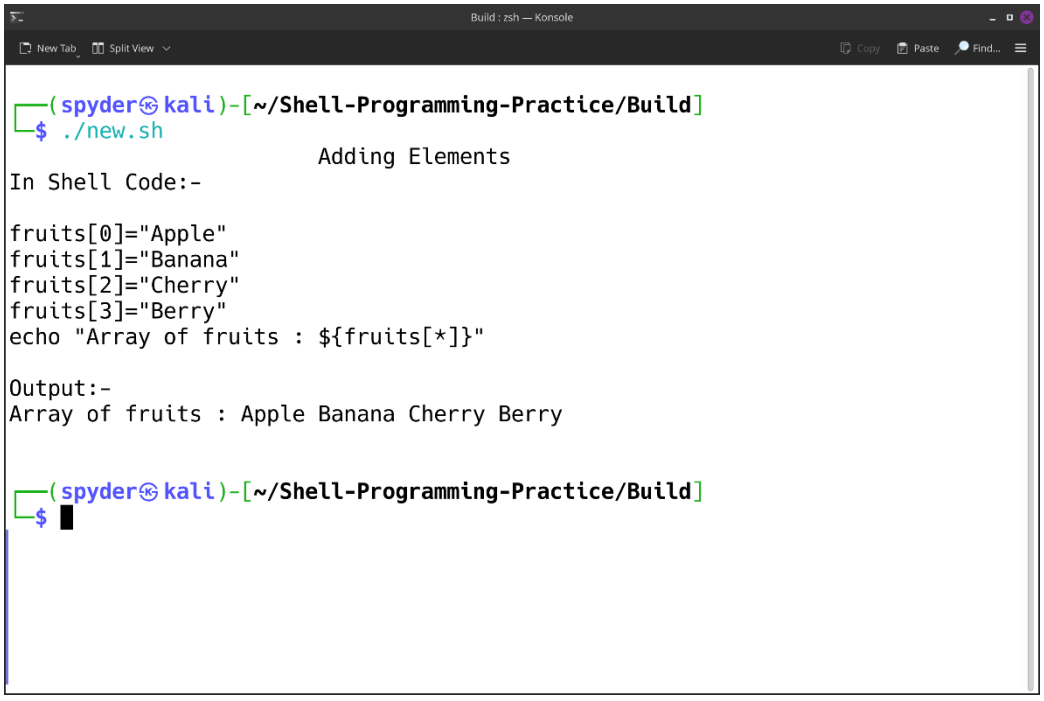

Fig 2.16 : Adding elements in Array

2.17 Array Length :

Code :

```
#!/bin/bash
printf "\t\t\tArray Length\n"
printf "In Shell Code:-\n\n"
echo 'echo "Size of fruits array : ${#fruits[*]}"'
printf "\nOutput:-\n"
fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"
echo "Size of fruits array : ${#fruits[*]}"
echo ""
```

Output :

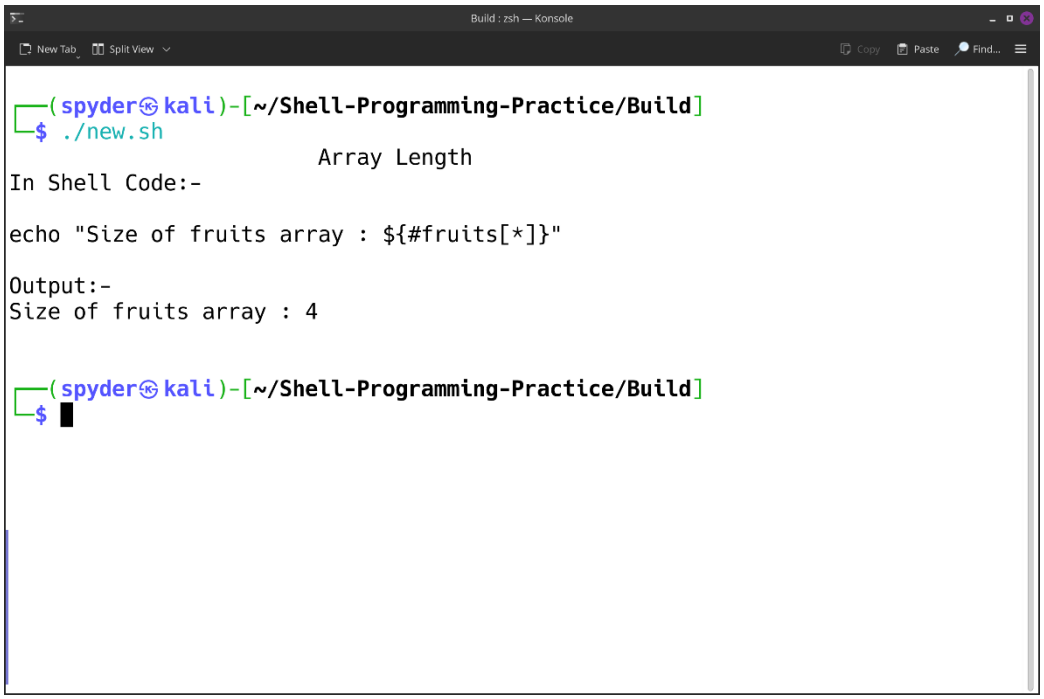

Fig 2.17 : Array length

2.18 Looping through Array :

Code :

```
#!/bin/bash
fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"
printf "\t\t\tLooping Through Array\n"
printf "In Shell Code:-\n\n"
echo 'for fruits in ${fruits[*]}
    do
        echo "$fruits"
    done'
printf "\nOutput:-\n"
for fruits in ${fruits[*]}
do
echo "$fruits"
done
echo ""
```

Output :

Fig 2.18 : looping through array

2.19 Update Array Element :

Code :

```
#!/bin/bash

fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"

printf "\t\t\tUpdate Array Element\n"
printf "In Shell Code:-\n\n"
echo 'fruits[1]="blueberry"
echo "Updated Array ${fruits[*]}"'

printf "\nOutput:-\n"
fruits[1]="blueberry"
echo "Updated Array : ${fruits[*]}"  # Output: apple blueberry
cherry
echo ""
```

Output :

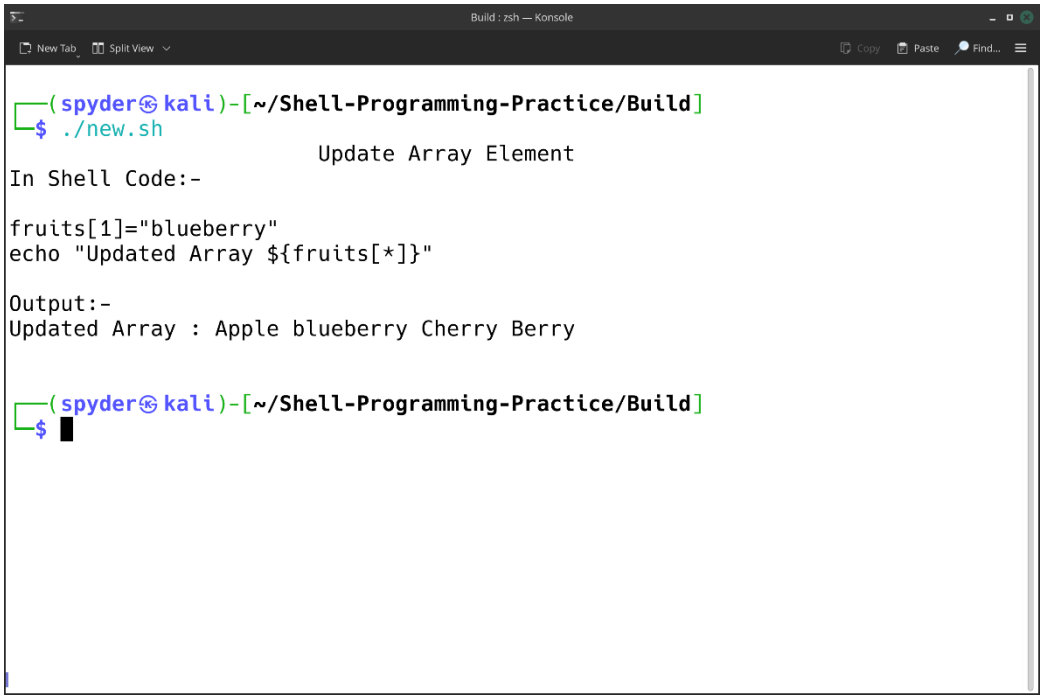

Fig 2.19 : Update element

2.20 Insert Element in Array :

Code :

```
#!/bin/bash

fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"

printf "\t\t\tInsert New Element\n"
printf "In Shell Code:-\n\n"
echo 'fruits+=($newElement) #OR
fruits+=("date")
echo ${fruits[*]}'
printf "\nOutput:-\n"
read -p "Enter New Element : " nw
fruits+=($nw)
echo ${fruits[*]}
echo ""
```

Output :

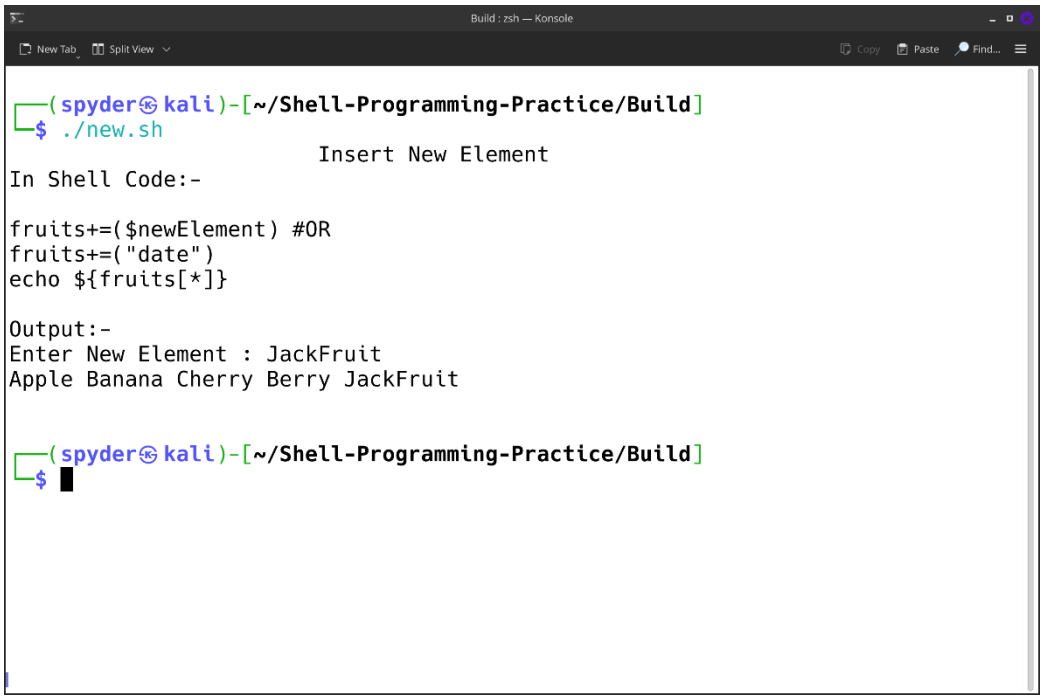

Fig 2.20 : Insert New Element

2.21 Delete Element in Array :

Code :

```
#!/bin/bash

fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"

printf "\t\t\tDelete Array Element\n"
printf "In Shell Code:-\n\n"
echo 'unset fruits[1]
echo ${fruits[*]}
'
printf "\nOutput:-\n"
echo "Array Elements : ${fruits[*]}"
echo "After Deleting fruits[1]"
unset fruits[1]
echo "Remaining Elements : ${fruits[*]}"
echo ""
```

Output :

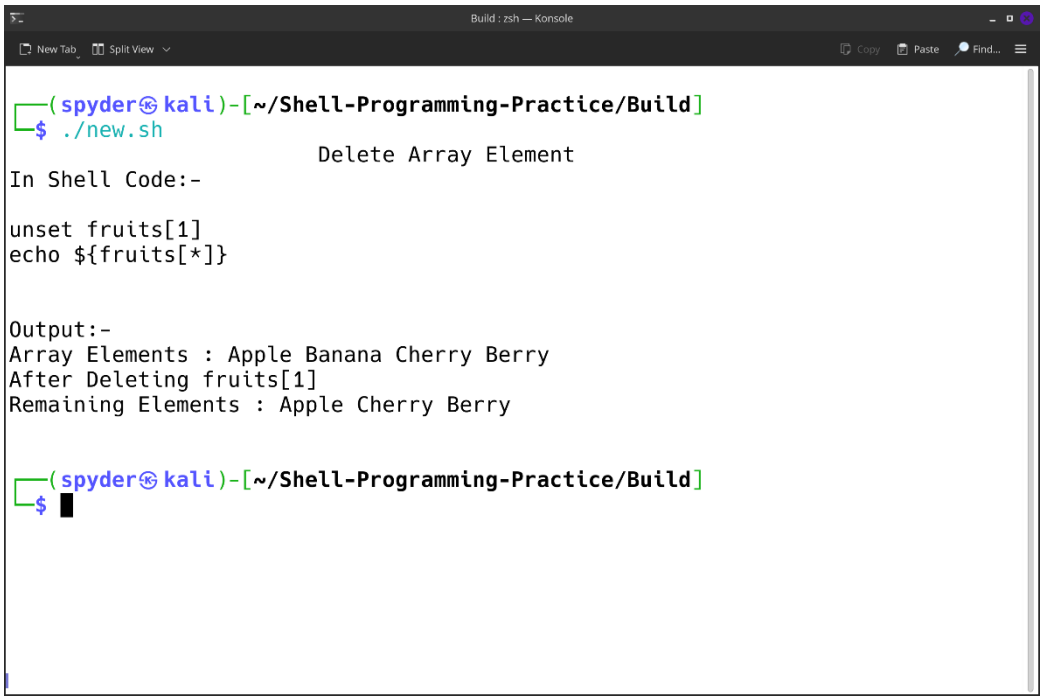

Fig 2.21 : Delete Element

2.22 Delete Entire Array :

Code :

```
#!/bin/bash

fruits[0]="Apple"
fruits[1]="Banana"
fruits[2]="Cherry"
fruits[3]="Berry"

printf "\t\t\tDelete Entire Array\n"
printf "In Shell Code:-\n\n"
echo 'unset fruits'
printf "\nOutput:-\n"
echo "Before Deleting Array : ${fruits[*]}"
echo "After Deleting Array :"
unset fruits
echo "array : ${fruits[*]}"
echo ""
```

Output :

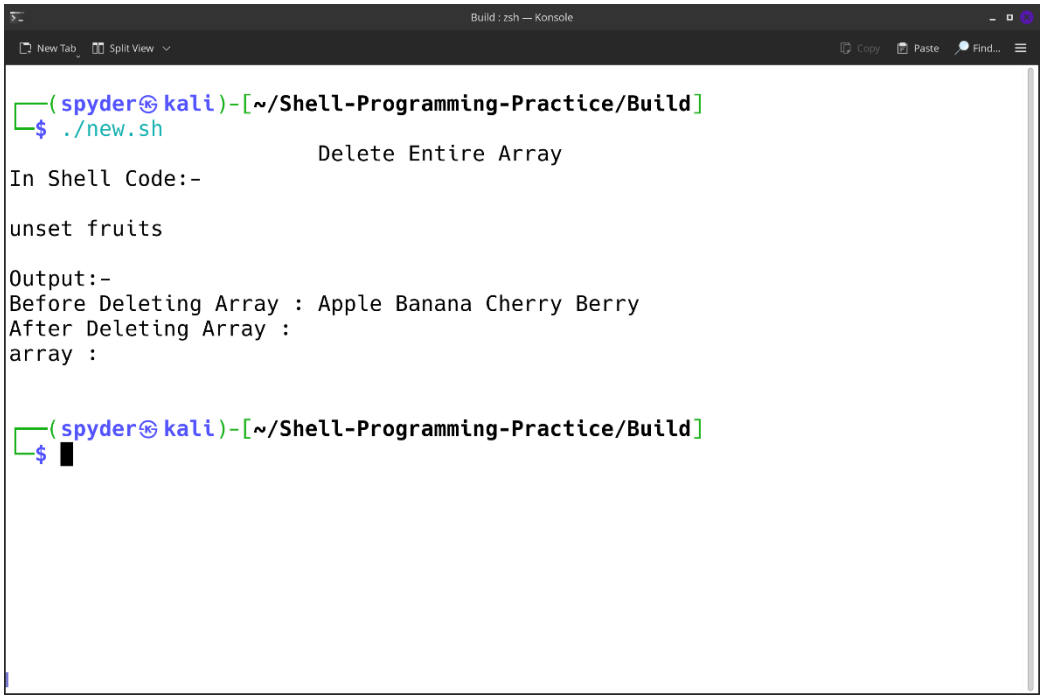

Fig 2.22 : Delete an Array

2.23 Declaring Associative Array :

Code :

```
#!/bin/bash

printf "\t\t\tDeclaring Associative Array\n"
printf "In Shell Code:-\n\n"
echo 'declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
'
printf "\nOutput:-\n"
declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
echo ""
echo "Person Array Elements : ${person[*]}"
echo ""
```

Output :

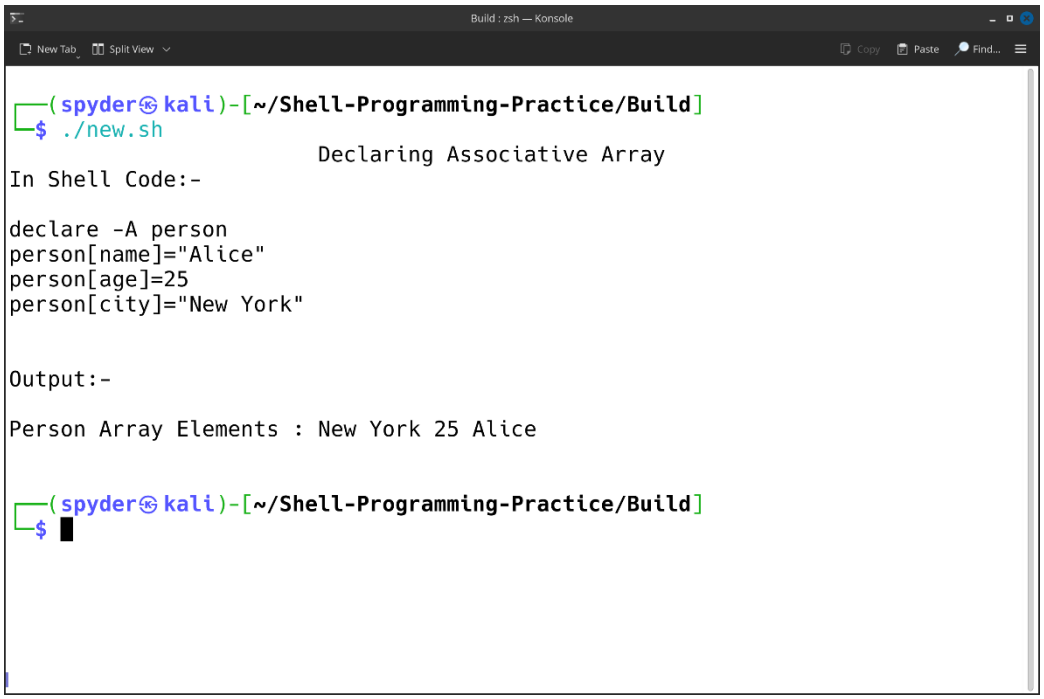

Fig 2.23 : Associative Array

2.24 Accessing Value by key in Associative array :

Code :

```
#!/bin/bash

printf "\t\t\tAccessing Value by key\n"
printf "In Shell Code:-\n\n"
echo 'echo ${person[name]}   # Output: Alice
echo ${person[city]}   # Output: New York'
printf "\nOutput:-\n"
declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
echo "key Name = ${person[name]}"   # Output: Alice
echo "key City = ${person[city]}"   # Output: New York
echo ""
```

Output :

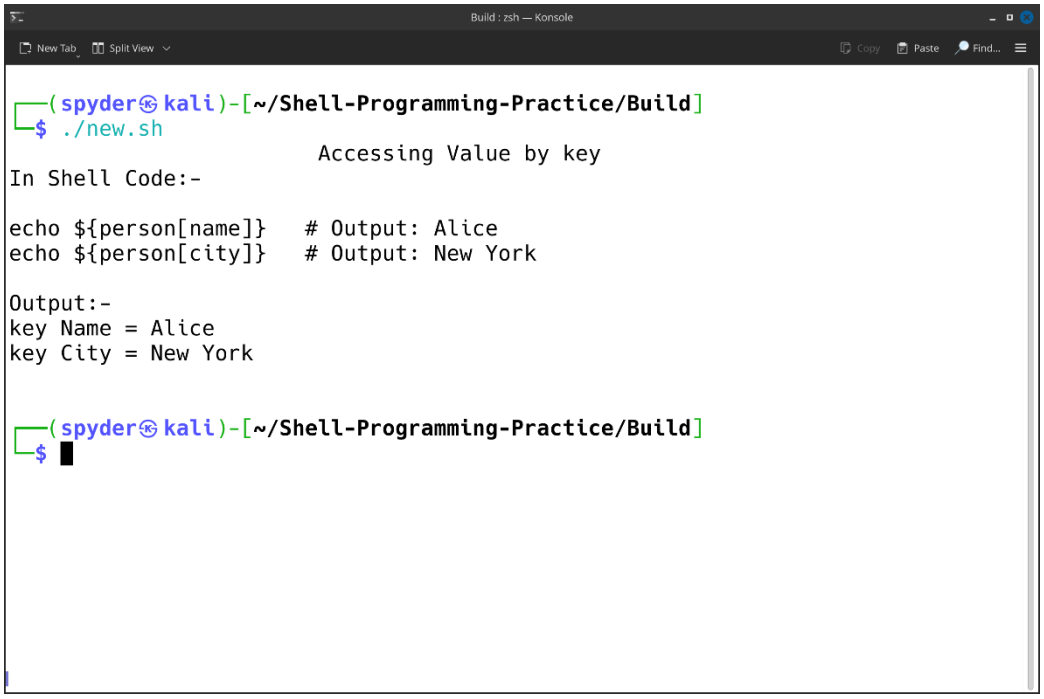

Fig 2.24 : Accessing value by Key

2.25 Looping through key in Associative array :

Code :

```
#!/bin/bash
printf "\t\t\tLooping Through Key Value\n"
printf "In Shell Code:-\n\n"
echo 'for key in "${!person[@]}"
do
  echo "$key: ${person[$key]}"
done
'
printf "\nOutput:-\n"
declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
for key in "${!person[@]}"
do
  echo "Person[$key] : ${person[$key]}"
done
echo ""
```

Output :

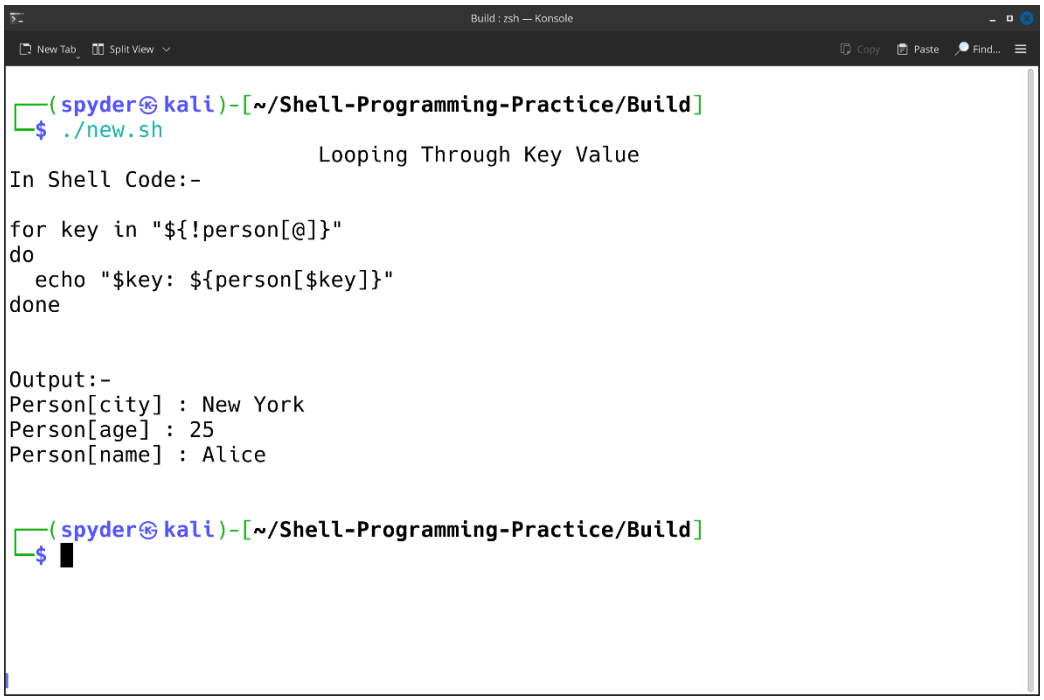

Fig 2.25 : Looping through Key value

2.26 Deleting from key in Associative array :

Code :

```
#!/bin/bash

printf "\t\t\tDeleting From key\n"
printf "In Shell Code:-\n\n"
echo 'unset person[age]'
printf "\nOutput:-\n"
declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
echo "Array Elements : ${person[*]}"
echo "After Deleting person[age]"
unset person[age]
echo "Remaining Elements : ${person[*]}"
echo ""
```

Output :

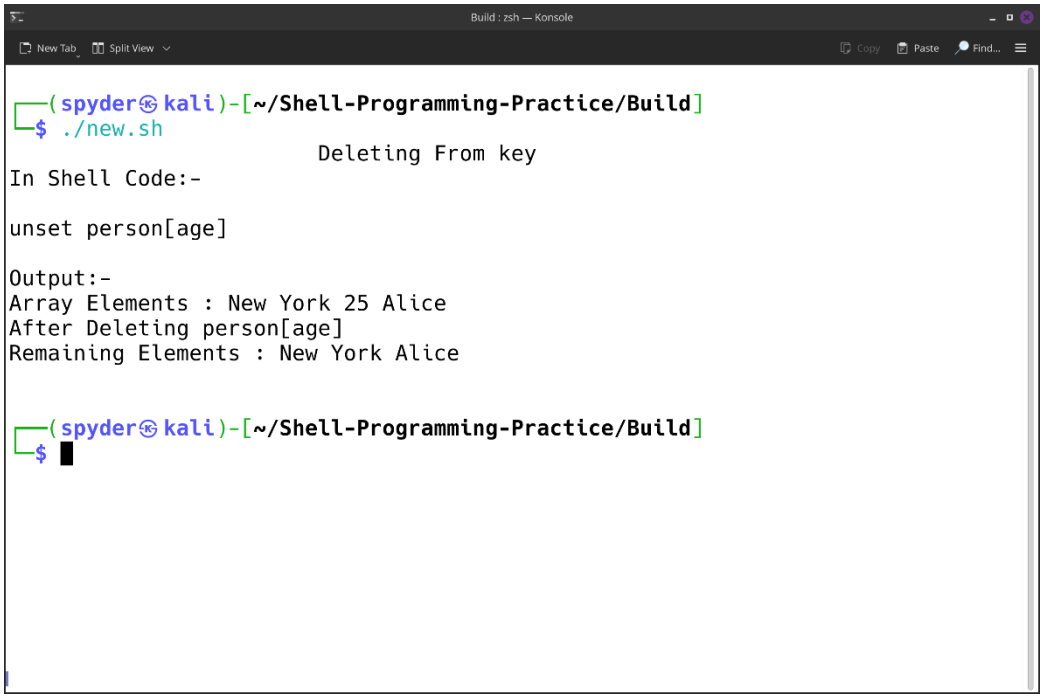

Fig 2.26 : Deleting From key

2.27 Display all key & values of Associative array :

Code :

```
#!/bin/bash

printf "\t\t\tDisplay all key & Values\n"
printf "In Shell Code:-\n\n"
echo 'echo ${!person[@]}   # Output: Show all keys
echo ${person[@]}   # Output: Show all values'
declare -A person
person[name]="Alice"
person[age]=25
person[city]="New York"
printf "\nOutput:-\n"
echo "All Keys are : ${!person[@]}"  # Output: Show all keys
echo "All Values are : ${person[@]}" # Output: Show all values
echo ""
```

Output :

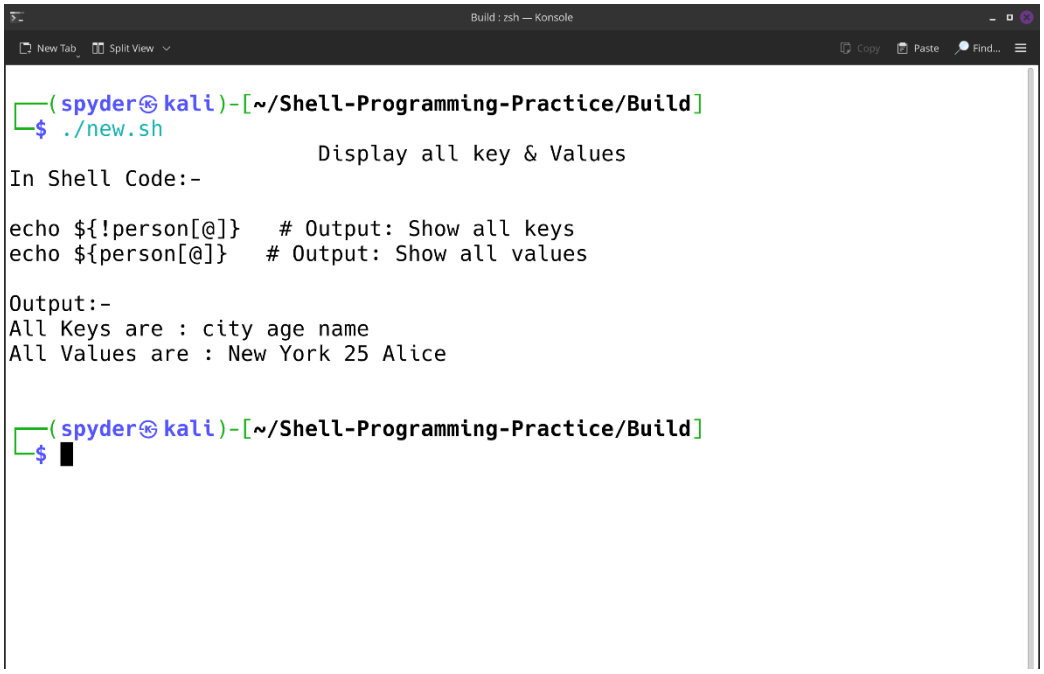

Fig 2.27 : Display all key & values