Elizabeth Aning-Amponsah 18412818

university of northampton

CSY2002 Operating systems

Assignment 2

|  |  |  |  |
| --- | --- | --- | --- |
| BSc Computing  **CSY2002 Operating Systems** | | | |
| Date of Issue: |  | Date for  Submission: | deadline Sunday 29 January 2023 |
| Agreed Date for late submission: | 13/06/2023 | Module Tutor:  Signed: | Ahmed Basil |
|  |
| Student Name:  Student ID: | Elizabeth Aning-Amponsah | | |
| 18412818 | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Student’s**  Signature: | **Elizabeth Aning-Amponsah** | | | | | |
| This assignment is weighted as 50% of the Module's assessment | | | | | | |
| **Assessment Feedback** | | | | | | |
| Aspect | | Excellent | Good | Satisfactory | Needs some more work | Needs much more work |
| area (20%) | |  |  |  |  |  |
| prompt (5%) | |  |  |  |  |  |
| bonus.sh (35%) | |  |  |  |  |  |
| blackjack.sh (40%) | |  |  |  |  |  |
|  | |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Specific aspects of the assignment that the marker likes: | | Specific aspects of the assignment that need more work: | | | |
| **Tutor’s** Signature: |  | Date: |  | Grade: |  |

The University of Northampton’s Policy on Plagiarism & Mitigating Circumstances will be strictly implemented. By submitting this **signed** assignment you are asserting that this submission is entirely your own individual work.

Table of Contents

[3. Area of Rectangle program 4](#_Toc137590754)

[Programme code 4](#_Toc137590755)

[Area script: flow chart 6](#_Toc137590756)

[Area script: user guide 7](#_Toc137590757)

[Area script: Code breakdown & description 8](#_Toc137590758)

[Area script: Testing 11](#_Toc137590759)

[Area script: Evidence of testing 12](#_Toc137590760)

[4. salesperson bonus programme 15](#_Toc137590761)

[Programme code 15](#_Toc137590762)

[Bonus script: Flowchart 20](#_Toc137590763)

[Bonus script: user guide 21](#_Toc137590764)

[Bonus script: code description 22](#_Toc137590765)

[Bonus Script: Testing 31](#_Toc137590766)

[Bonus Script: Evidence of testing 32](#_Toc137590767)

[5. Blackjack programme 36](#_Toc137590768)

[Programme code 36](#_Toc137590769)

[Blackjack script: flow chart 41](#_Toc137590770)

[Blackjack script: User guide 42](#_Toc137590771)

[Blackjack script: Code description 43](#_Toc137590772)

[Blackjack script: Testing 51](#_Toc137590773)

[Blackjack script: evidence of Testing 52](#_Toc137590774)

# 3. Area of Rectangle program

## Programme code

#!/bin/bash

# Create a variable for the constant

constant=2.54

#variable created to ensure valid entry

valid\_input=false

#while loop to ensure on 1 or 2 is entered

while [ "$valid\_input" = false ]; do

read -p "Please enter a unit of your choice to begin calculating an area of a rectangle [1 = cm, 2 = inch]: " enter\_unit

if [ "$enter\_unit" = "1" ] || [ "$enter\_unit" = "2" ]; then

valid\_input=true

else

echo "Invalid input! Please enter either 1 or 2."

fi

done

# Prompts user to enter value for both length and width

read -p "Enter a length of a rectangle: " length

read -p "Enter a width of a rectangle : " width

area=$(($length \* $width))

# Enables user to convert the value or remain the same

read -p "What unit do you want your rectangle area to display in [1 = cm, 2 = inch]: " final\_unit

# Displays the area in inches if they select 2

if [ "$final\_unit" == 2 ] && [ "$enter\_unit" == 2 ]; then

echo "The final area of the rectangle is $area inch"

# Displays the area in cm if they select 1

elif [ "$final\_unit" == 1 ] && [ "$enter\_unit" == 1 ]; then

echo "The final area of the rectangle is $area cm"

# Displays the final area converted from inches to cm

elif [ "$final\_unit" == 1 ] && [ "$enter\_unit" == 2 ]; then

INCHTOCM=$(echo "scale=3; $area \* $constant" | bc)

echo "The final area of the rectangle is $INCHTOCM cm"

# Displays the final area converted from cm to inches

elif [ "$final\_unit" == 2 ] && [ "$enter\_unit" == 1 ]; then

CMTOINCH=$(echo "scale=3; $area / $constant" | bc)

echo "The final area of the rectanggle is $CMTOINCH inch"

fi

#end of programme

read -p "Press 'Enter' to exit" end

## Area script: flow chart

A diagram of a flowchart

Description automatically generated with low confidence

## Area script: user guide

This is the user guide on how to navigate the area script program:

1. Run area.sh.
2. Enter digits 1 to display unit in cm or enter 2 to display in inches.
3. Enter length.
4. Enter width.
5. Enter digit 1 to change the display unit to cm or enter digit 2 to change the display unit to inches.
6. Display the area of the rectangle.
7. End

## Area script: Code breakdown & description

#!/bin/bash

# Create a variable for the constant

constant=2.54

in this area of the code the constant variable has been set to 2.54 as there are 2.54 centimeters in 1 inch enabling the conversion format in the following if/else statements.

#variable created to ensure valid entry

valid\_input=false

valid\_input variable is set to false to show that the user has not provided a valid input yet

#while loop to ensure on 1 or 2 is entered

while [ "$valid\_input" = false ]; do

read -p "Please enter a unit of your choice to begin calculating an area of a rectangle [1 = cm, 2 = inch]: " enter\_unit

if [ "$enter\_unit" = "1" ] || [ "$enter\_unit" = "2" ]; then

valid\_input=true

else

echo "Invalid input! Please enter either 1 or 2."

fi

done

The user is prompted to choose the unit they prefer, whether 1 = cm or 2 = inches. Whatever the user chooses will be stored in the enter\_unit variable.

The while loop ensures only 1 or 2 is selected to go ahead with the program, if any value other than 1 or 2 is selected, the error message “Invalid input! Please enter either 1 or 2.” Is displayed and prompts the user to enter the value again rather than quit the programme.

The if statement checks the value is valid and if valid\_input is set to true then the loop condition becomes false allowing the user to exit the loop and continue with the programme

# Prompts user to enter value for both length and width

read -p "Enter a length of a rectangle: " length

read -p "Enter a width of a rectangle : " width

area=$(($length \* $width))

length and width are stored in created value length and width variable once values have been input by user.

the area variable stores the equation to display an area which is length times width which the times symbol In this instance is the asterisk (\*).

# Enables user to convert the value or remain the same

read -p "What unit do you want your rectangle area to display in [1 = cm, 2 = inch]: " final\_unit

the final\_unit variable enables the user to change unit to cm from inches or inches from cm depending on what was selected at the first prompt of the program Or the user can let the unit remain the same by selecting the same unit as before.

# Displays the area in inches if they select 2

if [ "$final\_unit" == 2 ] && [ "$enter\_unit" == 2 ]; then

echo "The final area of the rectangle is $area inch"

this displays the area unit as inches if the user selected 2 and stored unit as inches in the enter\_unit variable and also selected 2 at the final\_unit variable and no changes are made to unit

# Displays the area in cm if they select 1

elif [ "$final\_unit" == 1 ] && [ "$enter\_unit" == 1 ]; then

echo "The final area of the rectangle is $area cm"

this displays the area unit as centimeters if the user selected 1 and stored unit as centimeters in the enter\_unit variable and also selected 1 at the final\_unit variable and no changes are made to unit.

# Displays the final area converted from inches to cm

elif [ "$final\_unit" == 1 ] && [ "$enter\_unit" == 2 ]; then

INCHTOCM=$(echo "scale=3; $area \* $constant" | bc)

echo "The final area of the rectangle is $INCHTOCM cm"

this displays the final area converted from inches (enter\_unit) to centimeters (final\_unit). To convert from inches to cm the unit in cm must be multiplied by 2.54 which in this instance is stored in the constant variable. Right after the multiplication in the code, it is followed by ‘bc’, which is used to represent a decimal number in bash scripting as bash only allows integer numbers and not double(decimal).

# Displays the final area converted from cm to inches

elif [ "$final\_unit" == 2 ] && [ "$enter\_unit" == 1 ]; then

CMTOINCH=$(echo "scale=3; $area / $constant" | bc)

echo "The final area of the rectanggle is $CMTOINCH inch"

fi

this displays the final area converted from centimeters (enter\_unit) to inches (final\_unit). To convert centimeters to inches the unit of centimeters which is the area in this instance must be divided by 2.54 which in this case is stored in the constant variable.

#end of programme

read -p "Press 'Enter' to exit" end

a variable end is created to ensure the exit of the program once the area has been displayed.

## Area script: Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Expected Output | Output | Pass/Fail |
| Enter a number other than 1 or 2 when prompted to select a unit of choice | The programme will display an error message, prompt the user to try again using 1 or 2 without ending programme | Programme presents error message and allows user to try again without exiting programme | PASS |
| Enter text at prompt when to select a unit of choice | The programme will display an error message, prompt the user to try again using 1 or 2 without ending programme | Programme presents error message and allows user to try again without exiting programme | PASS |
| Enter correct values when prompted to select a unit of choice | Programme successfully runs with no errors. Displays the final area unit. | The final area unit is displayed with no errors | PASS |
| Convert from inches to centimetres | The programme converts the enter\_unit value from inches to centimetres for the final\_unit | No errors, and enter\_unit value successfully changes the final\_unit from inches to centimetres | PASS |
| Convert from centimetres to inches | The programme converts the enter\_unit value from centimetres to inches for the final\_unit | No errors, and enter\_unit value successfully changes the final\_unit from centimetres to inches | PASS |

## Area script: Evidence of testing

1. Testing every other number but 1 and 2 at the prompt

A screenshot of a computer

Description automatically generated

1. Testing string entry at prompt

A screenshot of a computer

Description automatically generated with medium confidence

1. Testing correct values at prompt

A screenshot of a computer

Description automatically generated with medium confidence

1. Testing the conversion from inches to centimeters

A screenshot of a computer

Description automatically generated with medium confidence

1. Testing the conversion from centimeters to inches

A screenshot of a computer

Description automatically generated

# 4. salesperson bonus programme

## Programme code

#!/bin/bash

# Function to calculate the month

function input {

# User input month

echo "Enter the month (e.g., 1 = January):"

read month

echo "Month = $month"

# Perform checks on the entered month

if [ "$month" -gt 12 ] || [ "$month" -lt 1 ] || [ -z "$month" ]; then

echo "Please enter the correct value for month ($month). Refer to the example (e.g., 1 = January) and try again."

exit 1

fi

# Array to map numeric values to month names

months=("January" "February" "March" "April" "May" "June" "July" "August" "September" "October" "November" "December")

# Get the corresponding month name from the array

month\_name=${months[$month - 1]}

printf "\n"

# Display the month name

echo "Month: $month\_name"

# Check if a numeric value has been entered

if ! [[ $month =~ ^[0-9]+$ ]]; then

echo "Invalid month entered. Please enter a numeric value for the month."

exit 1

fi

printf "\n"

# User input salesperson

salesperson=""

while [ -z "$salesperson" ]; do

echo "Enter salesperson's name:"

read salesperson

# Check if the name entered is not an empty string

if [ -z "$salesperson" ]; then

echo "Please enter a salesperson's name."

fi

# check the salesperson's name is a string and not an integer input

if [[ $salesperson =~ [^[:alpha:]] ]]; then

echo "Invalid salesperson name entered. Please enter a valid string."

salesperson=""

fi

printf "\n"

done

# Input the sale amount

sales=""

while [ -z "$sales" ]; do

echo "Enter the sale amount for this month:"

read sales

# Check if the amount entered is not an empty string

if [ -z "$sales" ]; then

echo "Please enter an amount."

continue

fi

# check the sales amount is an integer value and not a string

if ! [[ $sales =~ ^[0-9]+(\.[0-9]+)?$ ]]; then

echo "Invalid sales amount entered. Please enter a valid number."

sales=""

fi

printf "\n"

done

# Input the filename

echo "Enter the desired filename:"

read FileName

# Check if the filename entered is not an empty string

if [ -z "$FileName" ]; then

echo "Please enter a filename and try again."

exit 0

fi

printf "\n"

# Display the inputs

echo "Month: $month"

echo "Salesperson: $salesperson"

echo "Sales amount: $sales"

echo "Filename: $FileName"

}

# Call the input function

input

#calculate the bonus

function bonus

{

if [ $sales -lt 200000 ]

then

bonus=0

echo "bonus= $bonus"

elif [ $sales -ge 200000 ] && test $sales -lt 300000

then

bonus=10000

echo "bonus= $bonus"

else

if [ $sales -ge 300000 ] && test $sales -lt 400000

then

bonus=15000

echo "bonus= $bonus"

else

if [ $sales -ge 400000 ] && test $sales -lt 500000

then

bonus=20000

echo "bonus= $bonus"

else

if [ $sales -ge 500000 ] && test $sales -lt 650000

then

bonus=25000

echo "bonus= $bonus"

else

if [ $sales -ge 650000 ]

then

bonus=30000

echo "bonus= $bonus"

fi

fi

fi

fi

fi

}

bonus

# function to calculate the salary

bonus=-1000

function taxCal

{

# Set basic salary and tax rate

basic\_salary=2000

basic\_rate\_lower=$(expr 12501 / 12) ; basic\_rate\_upper=$(expr 50000 / 12)

High\_Rate\_lower=$(expr 12501 / 12) ; High\_Rate\_upper=$(expr 150000 / 12)

# Calculate total monthly salary

Total\_monthly\_salary=$((basic\_salary + bonus))

# Check if total monthly salary falls within the basic rate

if [ $Total\_monthly\_salary -lt $basic\_rate\_lower ]

then

Tax=0

Salary\_after\_tax=$((Total\_monthly\_salary - Tax))

else

# Check if total monthly salary falls within the basic rate upper limit

if [ $Total\_monthly\_salary -le $basic\_rate\_upper ]

then

Tax=$(expr $basic\_rate\_upper / 20)

Salary\_after\_tax=$((Total\_monthly\_salary - Tax))

echo "Total salary after Tax = $Salary\_after\_tax"

else

# Check if total monthly salary falls within the high rate

test $Total\_monthly\_salary -gt $High\_Rate\_lower

Tax=$(expr $basic\_rate\_high / 40)

Salary\_after\_tax=$((Total\_monthly\_salary - Tax))

echo "Total salary after Tax = $Salary\_after\_tax"

fi

fi

}

taxCal

#check the correct input model

#!/bin/bash

function model {

printf "\n"

printf "\n"

#Display model key and model classes

echo "Enter the model of the car"

printf "Model key: "

printf "\n"

printf "#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+"

printf "\nA class - £24,095 - £38,095 (average £31,095)\nB Class - £28,045 - £38,280 (average £33,162)\nC Class - £34,670 - £50,405 (average £42,537)\nE Class - £39,680 - £69,015 (average £54,437)\nAMG C65 - £78,103 - £81,217 (average £79,660)"

printf "\n#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+"

printf "\n"

#prompt user to enter model key number associated with model class

printf "\nIf the sale falls under A class, enter 1\nIf the sale falls under B class, enter 2\nIf the sale falls under C class, enter 3\nIf the sale falls under E class, enter 4\nIf the sale is AMG C65, enter 5\nModel: "

read model

#check if key number associated to model class is valid

if [ "$model" -lt 1 ] || [ "$model" -gt 5 ]; then

echo "Please enter a number between 1 and 5 and try again"

checkModel=1

else

checkModel=0

case $model in

1) enteredModel="A class" ;;

2) enteredModel="B class" ;;

3) enteredModel="C class" ;;

4) enteredModel="E class" ;;

5) enteredModel="AMG\_C65 class" ;;

esac

echo "You entered Model = $enteredModel"

fi

#error message, prompt user to try again

while [ "$checkModel" -eq 1 ]; do

echo "You have entered the wrong value, please try again"

break

done

}

# Call the model function

model

# Function to write the results to a file

echo "The sales for month $months in 2023 is £$sales, and $salesperson has been paid £$Salary\_after\_tax in salary after tax" >> $FileName.txt

read -p "Enter to end" end

## Bonus script: Flowchart

A picture containing diagram, sketch, drawing, plan

Description automatically generated

## Bonus script: user guide

This is the user guide on how to navigate the bonus script program:

1. Run the programme bonus.sh
2. Enter month in digit format
3. Enter salespersons name
4. Enter the sale amount for that month
5. Enter filename
6. Displays all user inputs & bonus amount
7. Select the class the sale belongs to
8. Displays model class selected, saves to txt file.

## Bonus script: code description

# Function to calculate the month

function input {

# User input month

echo "Enter the month (e.g., 1 = January):"

read month

echo "Month = $month"

input function is created for all user inputs throughout this programme

the programme reads the month from a digit to a string which is stored in an array later in the code.

# Perform checks on the entered month

if [ "$month" -gt 12 ] || [ "$month" -lt 1 ] || [ -z "$month" ]; then

echo "Please enter the correct value for month ($month). Refer to the example (e.g., 1 = January) and try again."

exit 1

fi

the if statement ensures that the month variable is not empty, so user cannot enter empty data otherwise the error message displayed in the echo statement appears, it also checks that only a valid number is entered between 1 and 12 as they represent January to December followed by an exit 1 code which takes the user back to the beginning of the function within the script without quitting the programme altogether.

# Array to map numeric values to month names

months=("January" "February" "March" "April" "May" "June" "July" "August" "September" "October" "November" "December")

# Get the corresponding month name from the array

month\_name=${months[$month - 1]}

printf "\n"

# Display the month name

echo "Month: $month\_name"

The array stores the months in a string format.

The month name is retrieved from the stored array based on the user inputs value.

The subtraction of 1 from $month enables the index of the array to be properly situated so that if for example $month is 1, it the retrieves the element at index 0 which in this case is January as arrays always start from 0.

Printf “\n” is a new line that is empty

Whilst the echo statement prints out the month in a string format according to user input.

# Check if a numeric value has been entered

if ! [[ $month =~ ^[0-9]+$ ]]; then

echo "Invalid month entered. Please enter a numeric value for the month."

exit 1

fi

printf "\n"

This if statement uses a check to see if a numeric value has been entered and if not it takes the user back to the beginning of the function rather than quit altogether

The condition of the if statement ‘ [[ $month =~ ^[0-9]+$ ]]; ‘ checks to see if the value of $month (user input) does not match a numeric value and if it doesn’t, then the error message will appear

# User input salesperson

salesperson=""

while [ -z "$salesperson" ]; do

echo "Enter salesperson's name:"

read salesperson

# Check if the name entered is not an empty string

if [ -z "$salesperson" ]; then

echo "Please enter a salesperson's name."

fi

# check the salesperson's name is a string and not an integer input

if [[ $salesperson =~ [^[:alpha:]] ]]; then

echo "Invalid salesperson name entered. Please enter a valid string."

salesperson=""

fi

printf "\n"

done

this code is using two while loops. The first while loop is to ensure that the salesperson does not enter an empty value otherwise the echo error message will apper.

The second while loop ensures that the user inputs a string value and not an integer otherwise and echo error statement will appear and will prompt the use to enter a valid string.

# Input the sale amount

sales=""

while [ -z "$sales" ]; do

echo "Enter the sale amount for this month:"

read sales

# Check if the amount entered is not an empty string

if [ -z "$sales" ]; then

echo "Please enter an amount."

continue

fi

# check the sales amount is an integer value and not a string

if ! [[ $sales =~ ^[0-9]+(\.[0-9]+)?$ ]]; then

echo "Invalid sales amount entered. Please enter a valid number."

sales=""

fi

printf "\n"

done

this block of code ensures that the user inputs a valid integer value.

The while loop ensures that the $sales is not entered as an empty value without quiting the code.

The continue statement skips the rest of the components within the loop to proceed to the next.

Within the loop the user is prompted to enter a sales amount.

If the input is not empty, The following block of code ‘ [[ $sales =~ ^[0-9]+(\.[0-9]+)?$ ]]; ‘ is an expression that matches to a numeric value which essesntially is checking that the user has entered a numeric value, the user cannot proceed without entering a numeric value.

# Input the filename

echo "Enter the desired filename:"

read FileName

# Check if the filename entered is not an empty string

if [ -z "$FileName" ]; then

echo "Please enter a filename and try again."

exit 0

fi

printf "\n"

# Display the inputs

echo "Month: $month"

echo "Salesperson: $salesperson"

echo "Sales amount: $sales"

echo "Filename: $FileName"

}

# Call the input function

input

the first 2 lines of the code prompts the user to enter a filename of choice .

the following code, which is the if statement checks to see if the entry is empty or not, if it is it displays an error message and exits the programme altogether

the last echo statements are a display of all the users inputs on separate files

the input function is called to execute the whole input process.

#calculate the bonus

function bonus

{

if [ $sales -lt 200000 ]

then

bonus=0

echo "bonus= $bonus"

elif [ $sales -ge 200000 ] && test $sales -lt 300000

then

bonus=10000

echo "bonus= $bonus"

else

if [ $sales -ge 300000 ] && test $sales -lt 400000

then

bonus=15000

echo "bonus= $bonus"

else

if [ $sales -ge 400000 ] && test $sales -lt 500000

then

bonus=20000

echo "bonus= $bonus"

else

if [ $sales -ge 500000 ] && test $sales -lt 650000

then

bonus=25000

echo "bonus= $bonus"

else

if [ $sales -ge 650000 ]

then

bonus=30000

echo "bonus= $bonus"

fi

fi

fi

fi

fi

}

bonus

the function bonus is defined which will calculate the bonus amount based off the sales amount.

The first if statement means if sales is less than (-lt) 200000, then the bonus the salesperson receives is 0.

Whilst the next statement (elif) means if the sales amount is greater than or equal (-ge) to 200,000 and less than (-lt) 300,000 then the salesperson receives a bonus of 10,000.

The next statement states that if the sale amount is greater than or equal (-ge) to 300,000 but less than (-lt) 400,000 the salesperson receives a bonus of 15,000.

The next if statement states that if the sales amount is greater than or equal (-ge) to 400,00 but less than (-lt) 500,000 then the salesperson will receive a bonus of 20,000.

The next if statement indicates that if sales amount is greater than or equal to 500,000 but less than 650,000 the sales person receives a bonus of 25,000.

The final if statements states if sales amount is greater than or equal to 650,000 in sales then the bonus the salesperson receives is set to 30,000

The calculated bonus is then displayed (called in bonus)

# function to calculate the salary

bonus=-1000

function taxCal

{

# Set basic salary and tax rate

basic\_salary=2000

basic\_rate\_lower=$(expr 12501 / 12) ; basic\_rate\_upper=$(expr 50000 / 12)

High\_Rate\_lower=$(expr 12501 / 12) ; High\_Rate\_upper=$(expr 150000 / 12)

# Calculate total monthly salary

Total\_monthly\_salary=$((basic\_salary + bonus))

# Check if total monthly salary falls within the basic rate

if [ $Total\_monthly\_salary -lt $basic\_rate\_lower ]

then

Tax=0

Salary\_after\_tax=$((Total\_monthly\_salary - Tax))

else

# Check if total monthly salary falls within the basic rate upper limit

if [ $Total\_monthly\_salary -le $basic\_rate\_upper ]

then

Tax=$(expr $basic\_rate\_upper / 20)

Salary\_after\_tax=$((Total\_monthly\_salary - Tax))

echo "Total salary after Tax = $Salary\_after\_tax"

else

# Check if total monthly salary falls within the high rate

test $Total\_monthly\_salary -gt $High\_Rate\_lower

Tax=$(expr $basic\_rate\_high / 40)

Salary\_after\_tax=$((Total\_monthly\_salary - Tax))

echo "Total salary after Tax = $Salary\_after\_tax"

fi

fi

}

taxCal

this block of code defines the taxCal function (defined at beginning) of which calculates the monthly salary after tax deductions.

The basic\_salary variable is set to 2000 whilst the tax rates are set.

The total month salary is then calculated by adding the basic salary and bonus together

The function then proceeds with its first if statement which checks if the monthly salary falls within the basic rate range. If it does, the tax is set to 0 and the salary after tax is calculated by subtracting the tax from the monthly salary.

The following if statement checks if the total monthly salary falls above the basic rate by checking if it falls within the basic rate upper limit. If it does, the tax is calculated as a percentage of the basic\_rate\_upper limit and the salary after tax is then calculated.

The final statement checks if the total monthly salary is above the basic\_rate\_upper limit, which if it does it would fall within the high rate range. The tax is then calculcated as a percentage of the high rate, whilst salary after tax is calculated.

The calculated salary after tax is then displayed.

#check the correct input model

#!/bin/bash

function model {

printf "\n"

printf "\n"

#Display model key and model classes

echo "Enter the model of the car"

printf "Model key: "

printf "\n"

printf "#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+"

printf "\nA class - £24,095 - £38,095 (average £31,095)\nB Class - £28,045 - £38,280 (average £33,162)\nC Class - £34,670 - £50,405 (average £42,537)\nE Class - £39,680 - £69,015 (average £54,437)\nAMG C65 - £78,103 - £81,217 (average £79,660)"

printf "\n#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+"

printf "\n"

#prompt user to enter model key number associated with model class

printf "\nIf the sale falls under A class, enter 1\nIf the sale falls under B class, enter 2\nIf the sale falls under C class, enter 3\nIf the sale falls under E class, enter 4\nIf the sale is AMG C65, enter 5\nModel: "

read model

the function model is defined.

The printf statements display the stored key for the model classes, to improve readability and ensure the user knows what key is assigned to what model class.

The user is the prompted to enter the model answer according to the key provided in the printf statements

Read model seleted

#check if key number associated to model class is valid

if [ "$model" -lt 1 ] || [ "$model" -gt 5 ]; then

echo "Please enter a number between 1 and 5 and try again"

checkModel=1

else

checkModel=0

case $model in

1) enteredModel="A class" ;;

2) enteredModel="B class" ;;

3) enteredModel="C class" ;;

4) enteredModel="E class" ;;

5) enteredModel="AMG\_C65 class" ;;

esac

echo "You entered Model = $enteredModel"

fi

#error message, prompt user to try again

while [ "$checkModel" -eq 1 ]; do

echo "You have entered the wrong value, please try again"

break

done

}

# Call the model function

model

the if statements ensure that the number selected is only the numbers used in the key to identify the model class. If it is less than 1 or greater thana 5 then an error message will appear.

However if it is a valid entry between 1 and 5 then the associated key will match the the model class, then the entered model will be displayed in a string formart.

The while statements after the if statement checks if the $checkModel is still set to 1, if it is, it means that an inavalid model number has been entered and will prompt the user to try again by calling the model function again until the right value has been entered.

The loop will only break if a valid number has been entered.

Model function is called.

# Function to write the results to a file

echo "The sales for month $months in 2023 is £$sales, and $salesperson has been paid £$Salary\_after\_tax in salary after tax" >> $FileName.txt

read -p "Enter to end" end

this code writes the results to a file using a redirection operator which had been specified by the user when the user was prompted to create a filename. This file includes the sales information, salespersons name, and salary after tax.

“end” prompts user to enter to end the programme, which quits the programme

## Bonus Script: Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Expected Output | Output | Pass/Fail |
| Enter a number other than 1 to 12 for user input | The programme will display an error message, prompt the user to try again and the programme will end | Programme presents error message, prompts user to try again and ends programme | PASS |
| Enter a string for user input for month | Error message displayed, prompts user to use string before continuing | Programme presents error message, prompts user to try again and ends programme | PASS |
| Enter correct values (1-12) | The programme will successfully move on to the next function | Programme successfully moves on to next function | PASS |
| Enter an integer for salespersons | Programme will display error message and prompt user to enter valid string before continuing | Error message displayed, prompts user to use string before continuing | PASS |
| Enter a string for salesperson | The programme will successfully move on to the next function | Programme successfully moves on to next function | PASS |
| Enter a string for sales amount | Programme will display error message and prompt user to enter valid integer value before continuing | Error message displayed, prompts user to use integer before continuing | PASS |
| Displays results and user input in .txt file | Programme should produce a txt file that contains the user input and result of bonus | .txt file is produced but the content is incomplete as it doesn’t display bonus or the correct salary | FAIL |

## Bonus Script: Evidence of testing

1. Entering a number other than 1 – 12:

A screenshot of a computer

Description automatically generated

1. Entering a string instead of an integer

A screenshot of a computer

Description automatically generated

1. Enter correct values

A screenshot of a computer

Description automatically generated

1. Entering integer for salespersons

A screenshot of a computer

Description automatically generated with medium confidence

1. Entering string for salesperson

A screenshot of a computer

Description automatically generated

1. Entering string for sales amount

A screenshot of a computer

Description automatically generated with medium confidence

1. Display results in .txt

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

# Blackjack programme

## Programme code

#!/bin/bash

#The deck of cards key

#C = Clubs

#D = Diamonds

#H = Hearts

#S = Spades

FreshDeck=('CAce' 'C2' 'C3' 'C4' 'C5' 'C6' 'C7' 'C8' 'C9' 'C10' 'CJack' 'CQueen' 'CKing'

'DAce' 'D2' 'D3' 'D4' 'D5' 'D6' 'D7' 'D8' 'D9' 'D10' 'DJack' 'DQueen' 'DKing'

'HAce' 'H2' 'H3' 'H4' 'H5' 'H6' 'H7' 'H8' 'H9' 'H10' 'HJack' 'HQueen' 'HKing'

'SAce' 'S2' 'S3' 'S4' 'S5' 'S6' 'S7' 'S8' 'S9' 'S10' 'SJack' 'SQueen' 'SKing');

playercount=0

playervalue=0

playDeck=(${FreshDeck[@]})

#Function to get rid of any blank cards

SkipBlankCards () {

while [[ $selected == 0 && $randomnum < 52 ]]

do

randomnum=$(( $randomnum + 1 ))

selected=${playDeck[$randomnum]}

done

}

#Function to get the value of King, Queen, Jack or Ace cards

SpecialCardValue () {

if [[ ($long = "King") || ($long = "Queen") || ($long = "Jack") ]]

then

playervalue=$(echo "$playervalue+10"|bc)

elif [[ $long == "Ace" ]]

then

echo "Ace Selected, Choose 1 or 11"

read ace

if [[ $ace = '1' ]]

then

playervalue=$(echo "$playervalue+1"|bc)

else

playervalue=$(echo "$playervalue+11"|bc)

fi

else

short=${selected:1:2}

playervalue=$(echo "$playervalue+$short"|bc)

fi

}

#Function to pick a random card for user

DealCard () {

randomnum=$(( ( $RANDOM % 51 ) + 1 ))

selected=${playDeck[$randomnum]}

}

#Dealer (computers) hand

dealercount=0

dealervalue=0

DealersHand () {

DealCard

if [[ $selected == '0' ]]

then

#Calling Function SkipBlank

SkipBlankCards

fi

dealerhand[$dealercount]=$selected

playDeck[$randomnum]=0

dealercount=$(echo "$dealercount+1"|bc)

long=${selected:1:5}

#Special Cards value

if [[ ($long = "King") || ($long = "Queen") || ($long = "Jack") ]]

then

dealervalue=$(echo "$dealervalue+10"|bc)

#Ace

elif [[ $long == "Ace" ]]

then

#Dealer's Ace value choice

if [[ $dealervalue < '6' ]]

then

dealervalue=$(echo "$dealervalue+1"|bc)

else

dealervalue=$(echo "$dealervalue+11"|bc)

fi

else

short=${selected:1:2}

dealervalue=$(echo "$dealervalue+$short"|bc)

fi

}

#Function to display the winner

DisplayWinner () {

#shows if theres no winner

if [[ ($dealervalue > "21") && ($playervalue > "21") ]]

then

echo "BOTH BUST, NO WINNER"

#displays that the player(user) won if the dealer cards bust over 21

elif [[ ($dealervalue > "21") && ($playervalue < "22") ]]

then

echo "DEALER BUST, PLAYER WINS"

#displays that the dealer won if the player(user) cards bust over 21

elif [[ ($dealervalue < "22") && ($playervalue > "21") ]]

then

echo "PLAYER BUST, DEALER WINS"

#displays that the game is a draw - both player and dealer get 21

elif [[ ($dealervalue == "21") && ($playervalue == "21") || ($playervalue == $dealervalue) ]]

then

echo "DRAW"

#displays that the dealer won if the dealers cards at 21 but the user hasn't

elif [[ ($dealervalue == "21") && ($playervalue != "21") ]]

then

echo "DEALER WINS"

#displays that the player(user) won if the players cards at 21 and the dealer hasn't

elif [[ ($dealervalue != "21") && ($playervalue == "21") ]]

then

echo "PLAYER WINS"

#displays who won if they got a number that is closer to the number of 21

elif [[ ($dealervalue != "21") && ($playervalue != "21") && ($playervalue < "21") && ($playervalue < "21") ]]

then

if [[ $dealervalue > $playervalue ]]

then

echo "DEALER WINS"

elif [[ $dealervalue < $playervalue ]]

then

echo "PLAYER WINS"

fi

else

echo " "

fi

}

#main

#Run Dealer's Hand

DealersHand

DealersHand

if [[ $dealervalue < '10' ]]

then

DealersHand

DealersHand

elif [[ $dealervalue > "10" && $dealervalue < "17" ]]

then

DealersHand

fi

# guide and keys for the special cards

echo "#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+"

echo " H- Hearts; S- Spades; D- Diamonds; C- Clubs "

echo " 1 to Deal, 2 to end game"

echo "#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+"

while [ "$deal" != 2 ]

do

echo "Deal? 1-Yes 2-No"

read deal

if [[ $deal = 1 ]]

then

#PLAYERS CARDS

#Deal Card function

DealCard

if [[ $selected == '0' ]]

then

#Calling Function SkipBlank

SkipBlankCards

fi

displayHand[$playercount]=$selected

playDeck[$randomnum]=0

playercount=$(echo "$playercount+1"|bc)

echo "Picked: "$selected

long=${selected:1:5}

#Function for the Ace Value

SpecialCardValue

#displays the players value

echo "Value is "$playervalue

fi

# While loop done

done

echo "Your Hand "${displayHand[@]}

echo "Value is "$playervalue

echo " "

echo "Dealer Hand "${dealerhand[@]}

echo "Dealer Value "$dealervalue

echo " "

#Function to display Winner called

DisplayWinner

read -p "Enter to end" end

## Blackjack script: flow chart

A picture containing diagram, sketch, technical drawing, plan

Description automatically generated

## Blackjack script: User guide

1. Run programme blackjack.sh
2. Enter 1 to deal card or enter 2 to exit
3. If 1 is entered, value of users hand is shown after each round
4. When the user is happy with their result, 2 is pressed
5. Dealers hand is revealed
6. Winner is determined and displayed between dealer and user

## Blackjack script: Code description

#!/bin/bash

#The deck of cards key

#C = Clubs

#D = Diamonds

#H = Hearts

#S = Spades

FreshDeck=('CAce' 'C2' 'C3' 'C4' 'C5' 'C6' 'C7' 'C8' 'C9' 'C10' 'CJack' 'CQueen' 'CKing'

'DAce' 'D2' 'D3' 'D4' 'D5' 'D6' 'D7' 'D8' 'D9' 'D10' 'DJack' 'DQueen' 'DKing'

'HAce' 'H2' 'H3' 'H4' 'H5' 'H6' 'H7' 'H8' 'H9' 'H10' 'HJack' 'HQueen' 'HKing'

'SAce' 'S2' 'S3' 'S4' 'S5' 'S6' 'S7' 'S8' 'S9' 'S10' 'SJack' 'SQueen' 'SKing');

playercount=0

playervalue=0

playDeck=(${FreshDeck[@]})

the commented code explains what key represents what card within the deck of cards

the FreshDeck array is initialised with all cards in the deck which each card represents as a string in the array

The playercount variable and value is set to 0.

The playDeck array is initialised as a copy of the FreshDeck array which will be used to keep a track of the cards being played and dealt

#Function to get rid of any blank cards

SkipBlankCards () {

while [[ $selected == 0 && $randomnum < 52 ]]

do

randomnum=$(( $randomnum + 1 ))

selected=${playDeck[$randomnum]}

done

}

The SkipBlankCards function contains a while loop that repeatedly checks 2 condiotions: the variable $selected equals 0 and the variable $randomnum is less than 52.

Within this loop, the $randomnum variable is incremented by 1.

The $selected variable is assigned the value of the element at $randomnum in the playDeck array, which means the current card from the playDeck array is assigned to $selected.

This loop will continue until the $selected variable is not equal to 0 or when $randomnum gets to 52

#Function to get the value of King, Queen, Jack or Ace cards

SpecialCardValue () {

if [[ ($long = "King") || ($long = "Queen") || ($long = "Jack") ]]

then

playervalue=$(echo "$playervalue+10"|bc)

elif [[ $long == "Ace" ]]

then

echo "Ace Selected, Choose 1 or 11"

read ace

if [[ $ace = '1' ]]

then

playervalue=$(echo "$playervalue+1"|bc)

else

playervalue=$(echo "$playervalue+11"|bc)

fi

else

short=${selected:1:2}

playervalue=$(echo "$playervalue+$short"|bc)

fi

}

The SpecialCardValue is responsible for determining the value of the special cards e.g. King, Queen, Jack and Ace.

If one of the special cards: kings, queen and jack are selected within the game, the value will be incremented by 10 however if ace is selected the value has an option of being incremented by 1 or 11 of which the user permits it to be.

#Function to pick a random card for user

DealCard () {

randomnum=$(( ( $RANDOM % 51 ) + 1 ))

selected=${playDeck[$randomnum]}

}

The DealCard function is responsible for randomly selecting a card from the playDeck array and assigning it to the $selected variable. It does this by randomly generating numbers from 0 to 51 as there are only 52 elements within the array.

#Dealer (computers) hand

dealercount=0

dealervalue=0

DealersHand () {

DealCard

if [[ $selected == '0' ]]

then

#Calling Function SkipBlank

SkipBlankCards

fi

dealerhand[$dealercount]=$selected

playDeck[$randomnum]=0

dealercount=$(echo "$dealercount+1"|bc)

long=${selected:1:5}

the dealercount has been initialized to 0 to show how many cards the dealer starts with

the dealervalue has also been initialised to 0.

The DealersHand function is responsible for adding a card to the dealers hand whilst updating the variables and arrays.

The DealCard function is called which will randomly select a card and assign it to the user in the $selected variable

The function then checks if the value of $selected is 0, it is, it called the SkipBlankCards function as there are no blank cards in a blackjack game

Once the card has been dealt the card is then removed from the playDeck (playDeck[$randomnum]=0)

The dealercount variable is then incremented by 1 to reflect the added card in the dealers hand

The long variable card enables the card from the array to be removed and identify the card.

#Special Cards value

if [[ ($long = "King") || ($long = "Queen") || ($long = "Jack") ]]

then

dealervalue=$(echo "$dealervalue+10"|bc)

#Ace

elif [[ $long == "Ace" ]]

then

#Dealer's Ace value choice

if [[ $dealervalue < '6' ]]

then

dealervalue=$(echo "$dealervalue+1"|bc)

else

dealervalue=$(echo "$dealervalue+11"|bc)

fi

else

short=${selected:1:2}

dealervalue=$(echo "$dealervalue+$short"|bc)

fi

}

This code checks if the value of $long is either one of the special cards (king, queen or jack), if it is true then the dealers value is incremented by 10.

If the value of $long is ace, the programme prompts the user to select between incrementing by 1 or by 11

If the dealers current value is less than 6, the dealer chooses to count the ace as 1, otherwise the dealer would choose 11.

If none of the conditions are met then no special card is selected

Function to display the winner

DisplayWinner () {

#shows if theres no winner

if [[ ($dealervalue > "21") && ($playervalue > "21") ]]

then

echo "BOTH BUST, NO WINNER"

#displays that the player(user) won if the dealer cards bust over 21

elif [[ ($dealervalue > "21") && ($playervalue < "22") ]]

then

echo "DEALER BUST, PLAYER WINS"

#displays that the dealer won if the player(user) cards bust over 21

elif [[ ($dealervalue < "22") && ($playervalue > "21") ]]

then

echo "PLAYER BUST, DEALER WINS"

#displays that the game is a draw - both player and dealer get 21

elif [[ ($dealervalue == "21") && ($playervalue == "21") || ($playervalue == $dealervalue) ]]

then

echo "DRAW"

#displays that the dealer won if the dealers cards at 21 but the user hasn't

elif [[ ($dealervalue == "21") && ($playervalue != "21") ]]

then

echo "DEALER WINS"

#displays that the player(user) won if the players cards at 21 and the dealer hasn't

elif [[ ($dealervalue != "21") && ($playervalue == "21") ]]

then

echo "PLAYER WINS"

#displays who won if they got a number that is closer to the number of 21

elif [[ ($dealervalue != "21") && ($playervalue != "21") && ($playervalue < "21") && ($playervalue < "21") ]]

then

if [[ $dealervalue > $playervalue ]]

then

echo "DEALER WINS"

elif [[ $dealervalue < $playervalue ]]

then

echo "PLAYER WINS"

fi

else

echo " "

fi

}

This function determines and displays the winner of the game based on the values of the dealers hand ($dealervalue) and the players hand ($playersvalue)

If both the dealer and player’s hand are greater than 21, it means both have busted, to which the function outputs “BOTH BUST, NO WINNER”

If the dealers hand value is greater than 21 and the players hand value is less than or equal to 21, the dealer has busted and the player wins to which the function outputs “DEALER BUST, PLAYER WINS”

If the dealers hand value is less than or equal to 21 and the players hand value is greater than 21, it means the player has busted and the dealer wins to which the function outputs “PLAYER BUST, DEALER WINS”

If both dealer and players hand values are 21 or have the same value, it will draw and output “DRAW”

If the dealers hand is not 21 and the players hand value is 21, player wins and output is “PLAYER WINS”

If none of the conditions are met, it would mean that both values of the player and dealer are below 21. The function would then have to compare values to determine the winner based on who was closest to 21.

#Run Dealer's Hand

DealersHand

DealersHand

if [[ $dealervalue < '10' ]]

then

DealersHand

DealersHand

elif [[ $dealervalue > "10" && $dealervalue < "17" ]]

then

DealersHand

fi

the code starts by calling the DearlersHand function twice, this means 2 cards are dealt to the dealers hand.

After dealing the two cards, the code checks if the value of $dealervalue is less than 10 and if it is, the programme proceeds to call the DealersHand function two more times, meaning additional cards are dealt to the dealers hand

Whereas if the $dealervalue is greater than 10 and less than 17, the code calls the DealersHand function one more time meaning an additional card is also dealt to the dealers hand.

# guide and keys for the special cards

echo "#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+"

echo " H- Hearts; S- Spades; D- Diamonds; C- Clubs "

echo " 1 to Deal, 2 to end game"

echo "#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+#+"

while [ "$deal" != 2 ]

do

echo "Deal? 1-Yes 2-No"

read deal

if [[ $deal = 1 ]]

then

the echo statements are a guide to the special cards as well as an echo statement to give the user an option to play or quit the game.

The code enters a while loop which continues until the player decides to end the game by pressing ‘2’

Within the loop the code prompts the user with the message “Deal? 1-Yes 2-No” and reads the input into the $deal variable .

If the deal is 1 then the code proceeds to the next part of the script meaning the player has chosen to continue with the game

#PLAYERS CARDS

#Deal Card function

DealCard

if [[ $selected == '0' ]]

then

#Calling Function SkipBlank

SkipBlankCards

fi

displayHand[$playercount]=$selected

playDeck[$randomnum]=0

playercount=$(echo "$playercount+1"|bc)

echo "Picked: "$selected

long=${selected:1:5}

this code deals a card to the players hand and stores it in the displayHand array.

The code calls the DealCard function which selects a random card from the deck and assigns it to the $selected variable.

If the value of the $selected is equal to 0, it means a blank card was selected and in this case the code calls the SkipBlankCards function to skip the blank cards and selects the next available card.

Once it has been ensured that the selected card is not blank then the card is added to the players hand.

Once it has been added to the players hand, the playDeck array is then updated by removing the card from the deck whilst the $playercount variable is incremented by 1 to keep track of the cards in the players hand.

The selected card is then displayed with a message “picked: [selected card]

The long statement is used to determine the selection of a special card if selected and add its value to the players hand

#Function for the Ace Value

SpecialCardValue

#displays the players value

echo "Value is "$playervalue

fi

this code calls the SpecialCardValue function which calculates the value of special cards in the players hand and updates the $playervalue variable.

After updating the $playervalue a message will display the current value of the players hand

# While loop done

done

echo "Your Hand "${displayHand[@]}

echo "Value is "$playervalue

echo " "

echo "Dealer Hand "${dealerhand[@]}

echo "Dealer Value "$dealervalue

echo " "

#Function to display Winner called

DisplayWinner

read -p "Enter to end" end

this code displays the players hand, dealers nad their values and calls the ‘DisplayWinner’ function to determine the winner of the game.

The code then prompts the user to end which prevents the game from ending immediately after results have been displayed.

## Blackjack script: Testing

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Expected Output | Output | Pass/Fail |
| Enter a number other than 1 to 2 when prompted to deal | The programme will continuously ask the user to deal until integer value is selected | The programme has no errors and continues to ask user to deal | PASS |
| Enter a string for user input instead of a number to deal | The programme will continuously ask the user to deal until integer value is selected | The programme has no errors and continues to ask user to deal | PASS |
| Enter a value other than 1 or 11 for the ace | The programme will continuously ask the user to deal until integer value is selected | The programme has no errors and continues to ask user to deal | PASS |
| Enter the correct values | The programme will successfully with no errors | Programme ran successfully without errors | PASS |

## Blackjack script: evidence of Testing

1. Entered value other than 1 or 2 to deal

A screenshot of a computer

Description automatically generated with medium confidence

1. Entered string instead of an integer

A screenshot of a computer

Description automatically generated with medium confidence

1. Entered a value other than 1 or 11 at ace

A screenshot of a computer

Description automatically generated with medium confidence

1. Entered correct values

A picture containing text

Description automatically generated

A picture containing text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence