**1. Calculate the factorial of a given number.**

# Read number from user

echo "Enter a number:"

read num

# Initialize factorial to 1

factorial=1

# Calculate factorial using a loop

for (( i=1; i<=num; i++ ))

do

factorial=$((factorial \* i))

done

# Display the result

echo "Factorial of $num is $factorial"

**2.Perform arithmetic operations on a given integer and floating point number.**

# Read two numbers from the user

read -p "Enter the first number: " num1

read -p "Enter the second number: " num2

# Display menu of operations

echo "Select an operation to perform:"

echo "1. Addition"

echo "2. Subtraction"

echo "3. Multiplication"

echo "4. Division"

# Read the user's choice

read -p "Enter your choice (1-4): " choice

# Perform the chosen operation using bc

case $choice in

1)

result=$(echo "$num1 + $num2" | bc -l)

echo "Result of addition: $result"

;;

2)

result=$(echo "$num1 - $num2" | bc -l)

echo "Result of subtraction: $result"

;;

3)

result=$(echo "$num1 \* $num2" | bc -l)

echo "Result of multiplication: $result"

;;

4)

result=$(echo "$num1 / $num2" | bc -l)

echo "Result of division: $result"

;;

\*)

echo "Invalid choice. Please select a valid option (1-4)."

;;

Esac

**3. Shell script program to display file name, line count, word count and character count.**

# Prompt user to enter the file name

read -p "Enter the file name: " file\_name

# Use wc command to calculate line, word, and character counts

line\_count=$(wc -l < "$file\_name")

word\_count=$(wc -w < "$file\_name")

char\_count=$(wc -m < "$file\_name")

echo "File Name: $file\_name"

echo "Line Count: $line\_count"

echo "Word Count: $word\_count"

echo "Character Count: $char\_count"

**4. Shell script program to calculate the division obtained by the student if the marks for 5 different subjects are given and if division scheme is also given.**

# Function to calculate percentage and determine division

calculate\_division() {

total\_marks=$1

percentage=$(echo "scale=2; $total\_marks / 5" | bc) # Calculate percentage

# Determine division based on the scheme

if (( $(echo "$percentage >= 60" | bc -l) )); then

division="First Division"

elif (( $(echo "$percentage >= 50" | bc -l) )); then

division="Second Division"

elif (( $(echo "$percentage >= 40" | bc -l) )); then

division="Third Division"

else

division="Fail"

fi

echo "Total Marks: $total\_marks"

echo "Percentage: $percentage%"

echo "Division: $division"

}

# Input marks for 5 subjects

echo "Enter marks for 5 subjects (out of 100 each):"

read -p "Subject 1: " s1

read -p "Subject 2: " s2

read -p "Subject 3: " s3

read -p "Subject 4: " s4

read -p "Subject 5: " s5

total=$((s1 + s2 + s3 + s4 + s5))

calculate\_division $total

**5. Shell script program to display the total number of command line argument.**

#!/bin/bash

# Count the total number of command-line arguments

total\_arguments=$#

# Display the total count

echo "Total number of command-line arguments: $total\_arguments"

**6.Sequence Printing using Threads Problem: Write a program that creates three threads: Thread A, Thread B, and Thread C. The threads must print numbers in the following sequence: A1, B2, C3, A4, B5, C6 … upto 20 numbers. • Thread A prints A1, A4, A7, … • Thread B prints B2, B5, B8, … • Thread C prints C3, C6, C9, ... Requirements: • Use semaphores to control the order of execution of the threads. • Ensure no race conditions occur.**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#define MAX\_COUNT 20

// Define semaphores for controlling the order of execution

sem\_t sem\_a, sem\_b, sem\_c;

int count = 1;

// Thread A function

void\* thread\_a(void\* arg) {

while (count <= MAX\_COUNT) {

sem\_wait(&sem\_a); // Wait for permission to execute

if (count <= MAX\_COUNT && count % 3 == 1) {

printf("A%d\n", count);

count++;

}

sem\_post(&sem\_b); // Allow thread B to execute

}

return NULL;

}

// Thread B function

void\* thread\_b(void\* arg) {

while (count <= MAX\_COUNT) {

sem\_wait(&sem\_b); // Wait for permission to execute

if (count <= MAX\_COUNT && count % 3 == 2) {

printf("B%d\n", count);

count++;

}

sem\_post(&sem\_c); // Allow thread C to execute

}

return NULL;

}

// Thread C function

void\* thread\_c(void\* arg) {

while (count <= MAX\_COUNT) {

sem\_wait(&sem\_c); // Wait for permission to execute

if (count <= MAX\_COUNT && count % 3 == 0) {

printf("C%d\n", count);

count++;

}

sem\_post(&sem\_a); // Allow thread A to execute again

}

return NULL;

}

int main() {

pthread\_t thread1, thread2, thread3;

// Initialize semaphores

sem\_init(&sem\_a, 0, 1); // Thread A starts first

sem\_init(&sem\_b, 0, 0); // Thread B waits for A

sem\_init(&sem\_c, 0, 0); // Thread C waits for B

// Create threads

pthread\_create(&thread1, NULL, thread\_a, NULL);

pthread\_create(&thread2, NULL, thread\_b, NULL);

pthread\_create(&thread3, NULL, thread\_c, NULL);

// Wait for threads to finish

pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

pthread\_join(thread3, NULL);

sem\_destroy(&sem\_a);

sem\_destroy(&sem\_b);

sem\_destroy(&sem\_c);

return 0;

}

**7.Write a Shell script thar take internal mark, attendance percentage and external mark as input. The eligibility criteria for appearing the exam is given. If the condition is satisfied display the message "allowed for the semester"; otherwise print the message as per the ineligibility.**

# Input: Internal marks, Attendance percentage, External marks

read -p "Enter internal marks (out of 100): " internal\_marks

read -p "Enter attendance percentage: " attendance\_percentage

read -p "Enter external marks (out of 100): " external\_marks

# Eligibility criteria

internal\_min=35

attendance\_min=75

external\_min=35

# Check eligibility conditions

if [ "$internal\_marks" -ge "$internal\_min" ] && [ "$attendance\_percentage" -ge "$attendance\_min" ] && [ "$external\_marks" -ge "$external\_min" ]; then

echo "Allowed for the semester"

else

# Check for specific ineligibility reasons

if [ "$internal\_marks" -lt "$internal\_min" ]; then

echo "Not allowed for the semester: Internal marks less than $internal\_min."

elif [ "$attendance\_percentage" -lt "$attendance\_min" ]; then

echo "Not allowed for the semester: Attendance less than $attendance\_min%."

elif [ "$external\_marks" -lt "$external\_min" ]; then

echo "Not allowed for the semester: External marks less than $external\_min."

fi

fi

**8. Calculate the gross salary.**

#!/bin/bash

# Input: Basic salary, HRA, DA, Other allowances

read -p "Enter Basic Salary: " basic\_salary

read -p "Enter HRA percentage: " hra\_percentage

read -p "Enter DA percentage: " da\_percentage

read -p "Enter Other Allowances: " other\_allowances

# Calculate HRA and DA amounts

hra\_amount=$(echo "scale=2; $basic\_salary \* $hra\_percentage / 100" | bc)

da\_amount=$(echo "scale=2; $basic\_salary \* $da\_percentage / 100" | bc)

# Calculate Gross Salary

gross\_salary=$(echo "scale=2; $basic\_salary + $hra\_amount + $da\_amount + $other\_allowances" | bc)

# Display the Gross Salary

echo "Gross Salary: $gross\_salary"

**9. Calculate the SUM of digit of a given number.**

# Input: Read a number

read -p "Enter a number: " number

# Initialize sum to 0

sum=0

# Loop to calculate the sum of digits

while [ $number -gt 0 ]; do

digit=$((number % 10)) # Extract the last digit

sum=$((sum + digit)) # Add the digit to sum

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

done

# Output the sum of digits

echo "The sum of the digits is: $sum"

**10. Calculate the overtime pay of n employees according to the given constraints.**

# Input: Number of employees

read -p "Enter the number of employees: " n

# Input: Overtime pay rate

read -p "Enter the overtime pay rate (per hour): " overtime\_rate

# Loop to calculate overtime pay for each employee

for ((i = 1; i <= n; i++)); do

echo "Employee $i:"

read -p "Enter total hours worked: " hours\_worked

# Check if hours worked exceed 40

if [ "$hours\_worked" -gt 40 ]; then

# Calculate overtime hours and pay

overtime\_hours=$((hours\_worked - 40))

overtime\_pay=$((overtime\_hours \* overtime\_rate))

echo "Overtime hours: $overtime\_hours"

echo "Overtime pay: $overtime\_pay"

else

# If no overtime

echo "No overtime pay (worked 40 hours or less)."

fi

echo "" # Blank line for separation

done

**10. Write a C program that will create a child process to generate a Fibonacci series of specified length and store it in an array. The parent process will wait for the child to complete its task and then display the Fibonacci series and then display the prime Fibonacci number in the series along with its position with appropriate message.**#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdbool.h>

// Function to check if a number is prime

bool is\_prime(int num) {

if (num < 2) return false;

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) return false;

}

return true;

}

// Function to generate the Fibonacci series

void generate\_fibonacci(int fib[], int n) {

fib[0] = 0;

if (n > 1) {

fib[1] = 1;

for (int i = 2; i < n; i++) {

fib[i] = fib[i - 1] + fib[i - 2];

}

}

}

int main() {

pid\_t pid;

int n;

// Input the length of the Fibonacci series

printf("Enter the length of the Fibonacci series: ");

scanf("%d", &n);

if (n <= 0) {

printf("Invalid length. Please enter a positive integer.\n");

return 1;

}

int fib[n]; // Array to store the Fibonacci series

// Create a child process

pid = fork();

if (pid < 0) {

// Error in fork

perror("Fork failed");

return 1;

} else if (pid == 0) {

// Child process: Generate the Fibonacci series

generate\_fibonacci(fib, n);

// Write the Fibonacci series to a pipe (or shared memory can be used)

printf("Child: Fibonacci series generated.\n");

// Exit the child process

exit(0);

} else {

// Parent process: Wait for the child to complete

wait(NULL);

// Display the Fibonacci series

printf("Parent: Fibonacci series:\n");

for (int i = 0; i < n; i++) {

printf("%d ", fib[i]);

}

printf("\n");

// Find and display prime Fibonacci numbers and their positions

printf("Parent: Prime Fibonacci numbers and their positions:\n");

for (int i = 0; i < n; i++) {

if (is\_prime(fib[i])) {

printf("Fibonacci number %d at position %d is prime.\n", fib[i], i + 1);

}

}

}

return 0;

}

**PALINDROME**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdbool.h>

// Function to check if a number is a palindrome

bool is\_palindrome(int num) {

int original = num;

int reversed = 0;

while (num > 0) {

int digit = num % 10;

reversed = reversed \* 10 + digit;

num /= 10;

}

return original == reversed;

}

int main() {

pid\_t pid;

int number;

// Input the number

printf("Enter a number to check if it's a palindrome: ");

scanf("%d", &number);

// Create a child process

pid = fork();

if (pid < 0) {

// Error in fork

perror("Fork failed");

return 1;

} else if (pid == 0) {

// Child process: Check if the number is a palindrome

if (is\_palindrome(number)) {

printf("Child: %d is a palindrome.\n", number);

} else {

printf("Child: %d is not a palindrome.\n", number);

}

// Exit the child process

exit(0);

} else {

// Parent process: Wait for the child to complete

wait(NULL);

printf("Parent: Child process completed. Palindrome check is done.\n");

}

return 0;

}

**Executing script :-**

Toush script.sh

Chmod +x script.sh

./script.sh