**#!/bin/bash**

**# Check if a number is passed as argument**

**if [ -z "$1" ]; then**

**echo "Usage: $0 <number>"**

**exit 1**

**fi**

**n=$1**

**factorial=1**

**# Calculate factorial**

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

**factorial=$((factorial \* i))**

**done**

**echo "Factorial of $n is $factorial"**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <unistd.h>**

**#include <sys/types.h>**

**#include <sys/wait.h>**

**void reverseString(char\* str, char\* reversed) {**

**int len = strlen(str);**

**for (int i = 0; i < len; i++) {**

**reversed[i] = str[len - i - 1];**

**}**

**reversed[len] = '\0';**

**}**

**int isPalindrome(char\* str, char\* reversed) {**

**return strcmp(str, reversed) == 0;**

**}**

**int main() {**

**char input[100], reversed[100];**

**printf("Enter a string: ");**

**scanf("%s", input);**

**pid\_t pid = fork();**

**if (pid == 0) { // Child process**

**reverseString(input, reversed);**

**printf("Reversed string: %s\n", reversed);**

**exit(0);**

**} else if (pid > 0) { // Parent process**

**wait(NULL); // Wait for child process to complete**

**if (isPalindrome(input, reversed)) {**

**printf("The string is a palindrome.\n");**

**} else {**

**printf("The string is not a palindrome.\n");**

**}**

**} else {**

**perror("Fork failed");**

**return 1;**

**}**

**return 0;**

**}**

**#!/bin/bash**

**# Input marks for 5 subjects**

**echo "Enter marks for 5 subjects: "**

**read -p "Subject 1: " sub1**

**read -p "Subject 2: " sub2**

**read -p "Subject 3: " sub3**

**read -p "Subject 4: " sub4**

**read -p "Subject 5: " sub5**

**# Calculate total and percentage**

**total=$((sub1 + sub2 + sub3 + sub4 + sub5))**

**percentage=$((total / 5))**

**# Determine division**

**if [ $percentage -ge 60 ]; then**

**division="First division"**

**elif [ $percentage -ge 50 ]; then**

**division="Second division"**

**elif [ $percentage -ge 40 ]; then**

**division="Third division"**

**else**

**division="Fail"**

**fi**

**# Display result**

**echo "Total Marks: $total"**

**echo "Percentage: $percentage%"**

**echo "Division: $division"**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <pthread.h>**

**#include <semaphore.h>**

**// Semaphores for synchronization**

**sem\_t sem\_A, sem\_B, sem\_C;**

**// Shared variable to keep track of the current number**

**int current\_number = 1;**

**// Function for Thread A**

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

**while (current\_number <= 20) {**

**sem\_wait(&sem\_A); // Wait for Thread A's turn**

**if (current\_number <= 20) {**

**printf("A%d\n", current\_number);**

**current\_number++;**

**}**

**sem\_post(&sem\_B); // Signal Thread B to execute**

**}**

**pthread\_exit(NULL);**

**}**

**// Function for Thread B**

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

**while (current\_number <= 20) {**

**sem\_wait(&sem\_B); // Wait for Thread B's turn**

**if (current\_number <= 20) {**

**printf("B%d\n", current\_number);**

**current\_number++;**

**}**

**sem\_post(&sem\_C); // Signal Thread C to execute**

**}**

**pthread\_exit(NULL);**

**}**

**// Function for Thread C**

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

**while (current\_number <= 20) {**

**sem\_wait(&sem\_C); // Wait for Thread C's turn**

**if (current\_number <= 20) {**

**printf("C%d\n", current\_number);**

**current\_number++;**

**}**

**sem\_post(&sem\_A); // Signal Thread A to execute**

**}**

**pthread\_exit(NULL);**

**}**

**int main() {**

**// Initialize semaphores**

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

**sem\_init(&sem\_B, 0, 0); // Thread B starts blocked**

**sem\_init(&sem\_C, 0, 0); // Thread C starts blocked**

**// Create threads**

**pthread\_t threadA, threadB, threadC;**

**pthread\_create(&threadA, NULL, thread\_A, NULL);**

**pthread\_create(&threadB, NULL, thread\_B, NULL);**

**pthread\_create(&threadC, NULL, thread\_C, NULL);**

**// Wait for threads to finish**

**pthread\_join(threadA, NULL);**

**pthread\_join(threadB, NULL);**

**pthread\_join(threadC, NULL);**

**// Destroy semaphores**

**sem\_destroy(&sem\_A);**

**sem\_destroy(&sem\_B);**

**sem\_destroy(&sem\_C);**

**printf("Sequence printing program completed.\n");**

**return 0;**

**}**

#####################################

gedit sequence\_printing.c

gcc -pthread sequence\_printing.c

./a.out