**#!/bin/bash**

**rate\_per\_hour=12**

**for i in {1..10}; do**

**read -p "Enter hours worked by employee $i: " hours\_worked**

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

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

**overtime\_pay=$((overtime \* rate\_per\_hour))**

**else**

**overtime\_pay=0**

**fi**

**echo "Overtime pay for employee $i: Rs. $overtime\_pay"**

**done**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <unistd.h>**

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

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

**// Function to convert number to binary string**

**void toBinary(int n, char\* binary) {**

**int index = 0;**

**while (n > 0) {**

**binary[index++] = (n % 2) + '0';**

**n /= 2;**

**}**

**binary[index] = '\0';**

**// Reverse the string to get proper binary representation**

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

**for (int i = 0; i < len / 2; i++) {**

**char temp = binary[i];**

**binary[i] = binary[len - i - 1];**

**binary[len - i - 1] = temp;**

**}**

**}**

**// Function to check if a binary string is a palindrome**

**int isPalindrome(char\* binary) {**

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

**for (int i = 0; i < len / 2; i++) {**

**if (binary[i] != binary[len - i - 1]) {**

**return 0;**

**}**

**}**

**return 1;**

**}**

**int main() {**

**int n;**

**char binary[100];**

**printf("Enter a non-negative integer: ");**

**scanf("%d", &n);**

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

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

**toBinary(n, binary);**

**printf("Binary representation: %s\n", binary);**

**exit(0);**

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

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

**toBinary(n, binary); // Generate binary in parent as well**

**if (isPalindrome(binary)) {**

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

**} else {**

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

**}**

**} else {**

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

**return 1;**

**}**

**return 0;**

**}**

**#!/bin/bash**

**# Read inputs**

**read -p "Enter Internal marks (out of 20): " internal\_marks**

**read -p "Enter Attendance percentage: " attendance**

**read -p "Enter External marks (out of 50): " external\_marks**

**# Initialize variables**

**eligible=true**

**message=""**

**# Check criteria**

**if [ $internal\_marks -lt 16 ]; then**

**eligible=false**

**message+="Internal marks are insufficient. "**

**fi**

**if [ $attendance -lt 40 ]; then**

**eligible=false**

**message+="Attendance percentage is insufficient. "**

**fi**

**if [ $external\_marks -lt 24 ]; then**

**eligible=false**

**message+="External marks are insufficient. "**

**fi**

**# Display result**

**if [ $eligible == true ]; then**

**echo "Allowed for Next Semester."**

**else**

**echo $message**

**fi**

**#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