########STRING OPERATIONS

```
#!/bin/bash

# Check if two strings are equal or not
echo "Enter the first word:"
read str1
echo "Enter the second word:"
read str2

if [ "$str1" == "$str2" ]; then
   echo "Strings are equal"
else
   echo "Strings are not equal"
fi

# Concatenation
concat="$str1$str2"
echo "The concatenated string is $concat"

# Finding the length of a string
length=$(expr length "$concat")
echo "The length of the string is $length"

# Printing the characters at odd positions
echo "Characters at odd positions in the second word:"
for (( i = 0; i < ${#str2}; i += 2 )); do
   echo "${str2:i:1}"
done

# To print reverse of the string
reverse=""
for (( i = $length - 1; i >= 0; i-- )); do
   reverse="$reverse${concat:$i:1}"
done
echo "Reversed string is $reverse"

# To check if a given word is palindrome or not
echo "Enter the word:"
read str
reverse=""
for (( i = ${#str} - 1; i >= 0; i-- )); do
   reverse="$reverse${str:$i:1}"
done
```

```
if [ "$str" == "$reverse" ]; then
   echo "It is a palindrome word"
else
   echo "It is not a palindrome word"
fi

# To find the occurrence of a character in a string
echo "Enter a string:"
read st
echo "Enter the character you want to search for:"
read char

# Use grep to find the occurrence of the character in the string
count=$(echo "$st" | grep -o "$char" | wc -l)

echo "The character '$char' appears $count times in the string '$st'"
```

########ARITHMETIC OPERATIONS######

```
#!/bin/bash

# Basic Operations
echo "Enter two numbers:"
read a
read b

# Perform arithmetic operations
val=$(echo "$a + $b" | bc)
echo "The sum is $val"

echo "Addition of entered numbers is: $(expr $a + $b)"
echo "Subtraction of entered numbers is: $(expr $a - $b)"
echo "Multiplication of entered numbers is: $(expr $a \* $b)"
echo "Division of entered numbers is: $(expr $a / $b)"
echo ""

# Area and perimeter of circle
echo "Enter radius of circle:"
read r
echo "Area of circle is: $(echo "3.14 * $r * $r" | bc) sq. units"
echo "Perimeter of the circle is: $(echo "2 * 3.14 * $r" | bc) units"
echo ""
```

```
# Gross salary
echo "Enter basic salary:"
read sal
echo "Gross salary is Rs. $(echo "0.0165 * $sal + 0.003 * $sal + $sal" | bc)"
echo ""

# Mean salary if basic salary is 1200, 1460, 1356, 1800
val=$(echo "1200 + 1460 + 1356 + 1800" | bc)
val=$(echo "$val / 4" | bc)
echo "The mean salary is $val"
```

Aim: Implement C program demonstrate Shortest Remaining Job algorithm

Code –

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;
struct Process {
int id;
int arrivalTime;
int burstTime;
int completionTime;
int turnaroundTime;
int waitingTime;
};

bool compareArrivalTime(const Process& p1, const Process& p2) {
return p1.arrivalTime < p2.arrivalTime;
}

bool compareBurstTime(const Process& p1, const Process& p2) {
return p1.burstTime < p2.burstTime;
}

void calculateCompletionTime(vector<Process>& processes) {
int currentTime = 0;
for (int i = 0; i < processes.size(); ++i) {
currentTime = max(currentTime, processes[i].arrivalTime);
processes[i].completionTime = currentTime + processes[i].burstTime;
processes[i].turnaroundTime = processes[i].completionTime -
```

```
processes[i].arrivalTime; processes[i].waitingTime = processes[i].turnaroundTime -
processes[i].burstTime; currentTime = processes[i].completionTime;
}
}

void printProcesses(const vector<Process>& processes) {
cout
<<"Process\tArrivalTime\tBurstTime\tCompletionTime\tTurnaroundTime\tWaitingTime\n"; for
(const auto& process : processes) {
cout << process.id << "\t\t" << process.arrivalTime << "\t\t" << process.burstTime << "\t\t"
<< process.completionTime << "\t\t" << process.turnaroundTime << "\t\t" <<
process.waitingTime << "\n";
}
}

int main() {

int n;
cout << "Enter the number of processes: ";
cin >> n;

vector<Process> processes(n);
cout << "Enter arrival time and burst time for each
process:\n"; for (int i = 0; i < n; ++i) {
processes[i].id = i + 1;
cout << "Process " << i + 1 << " arrival time: ";
cin >> processes[i].arrivalTime;
cout << "Process " << i + 1 << " burst time: ";
cin >> processes[i].burstTime;
}

sort(processes.begin(), processes.end(),
compareArrivalTime); calculateCompletionTime(processes);
sort(processes.begin(), processes.end(), compareBurstTime);

cout << "\nSJF Non-preemptive
Scheduling:\n"; printProcesses(processes);

double totalTurnaroundTime = 0;
double totalWaitingTime = 0;
for (const auto& process : processes) {
totalTurnaroundTime += process.turnaroundTime;

totalWaitingTime += process.waitingTime;
```

```
}
double avgTurnaroundTime = totalTurnaroundTime / n;
double avgWaitingTime = totalWaitingTime / n;
cout << "\nAverage Turnaround Time: " << avgTurnaroundTime << endl;
cout << "Average Waiting Time: " << avgWaitingTime << endl;

return 0;
}
```

Aim: Implement a C++ program to demonstrate the FCFS algorithm.

```
#include <iostream>
#include <algorithm>
using namespace std;
struct Process {
int id;
int arrivalTime;
int burstTime;

};
void calculateTimes(int n, Process processes[], int completion[], int waiting[],
int turnaround[]) {
// Initialize completion time and waiting time for the first
process completion[0] = processes[0].arrivalTime +
processes[0].burstTime; waiting[0] = 0;
// Calculate completion, waiting, and turnaround times for the rest of the
processes
for (int i = 1; i < n; i++) {
// Completion time is the maximum of either the previous process
completion time or the arrival time
completion[i] = max(completion[i - 1], processes[i].arrivalTime)
+ processes[i].burstTime;
// Waiting time is the difference between completion time and burst
time waiting[i] = completion[i - 1] - processes[i].arrivalTime;
}
// Calculate turnaround times
for (int i = 0; i < n; i++) {
turnaround[i] = completion[i] - processes[i].arrivalTime;
}
}
void selectionSort(int n, Process processes[]) {
// Check if all arrival times are equal or zero
```

```
bool allEqual = true;
for (int i = 1; i < n; ++i) {
if (processes[i].arrivalTime != processes[0].arrivalTime) {
allEqual = false;
break;
}
}
// If all arrival times are equal or zero, sort processes based on burst time
using selection sort
if (allEqual) {
for (int i = 0; i < n - 1; i++) {
int minIndex = i;
for (int j = i + 1; j < n; j++) {
if (processes[j].burstTime < processes[minIndex].burstTime)
{ minIndex = j;
}
}
// Swap processes
swap(processes[i], processes[minIndex]);
}

}
}
int main() {
int n;
// Input number of processes
cout << "Enter the number of processes: ";
cin >> n;
Process processes[n];
int completion[n], waiting[n], turnaround[n];
// Input arrival time and burst time for each process
for (int i = 0; i < n; i++) {
processes[i].id = i + 1;
cout << "Enter arrival time for process " << i + 1 << ": ";
cin >> processes[i].arrivalTime;
cout << "Enter burst time for process " << i + 1 << ": ";
cin >> processes[i].burstTime;
}
// Sort processes based on arrival time if not all equal, otherwise sort
based on burst time
selectionSort(n, processes);
// Calculate completion, waiting, and turnaround times
calculateTimes(n, processes, completion, waiting, turnaround);
// Display results
```

```
cout << "\nProcess\t Arrival Time\t Burst Time\t Completion Time\t Waiting
Time\t Turnaround Time\n";
for (int i = 0; i < n; i++) {
cout << processes[i].id << "\t\t" << processes[i].arrivalTime <<
"\t\t" << processes[i].burstTime << "\t\t" << completion[i] << "\t\t" <<
waiting[i] << "\t\t" << turnaround[i] << endl;
}
// Calculate average waiting time and average turnaround
time float avgWaitingTime = 0, avgTurnaroundTime = 0;
for (int i = 0; i < n; i++) {
avgWaitingTime += waiting[i];
avgTurnaroundTime += turnaround[i];
}
avgWaitingTime /= n;
avgTurnaroundTime /= n;

cout << "\nAverage Waiting Time: " << avgWaitingTime << endl;
cout << "Average Turnaround Time: " << avgTurnaroundTime << endl;
return 0;
}
```

########Shell scripts for process operations

```
#!/bin/bash

# Function to list all processes
list_processes(){
   echo "Listing all processes:"
   ps
}

# Function to display information about a specific process
process_info(){
   read -p "Enter the PID of the process: " pid
   echo "Information about process $pid:"
   ps -p $pid
}

# Function to display the global priority of a specific process
global_priority(){
   read -p "Enter PID of process: " pid
   echo "Global priority of process $pid:"
   ps -p $pid -o pid,pri
}
```

```
# Function to change the priority of a specific process
change_priority(){
    read -p "Enter PID of process: " pid
    read -p "Enter new priority (default: 10): " priority
    priority=${priority:-10}
    renice $priority $pid
    echo "Priority of process $pid changed to $priority"
}

# Infinite loop for menu options
while true; do
    echo "1. List Processes"
    echo "2. Process Info"
    echo "3. Global Priority"
    echo "4. Change Priority (default: 10)"
    echo "5. Exit"
    read -p "Choose an option: " choice

    case $choice in
        1) list_processes ;;
        2) process_info ;;
        3) global_priority ;;
        4) change_priority ;;
        5) echo "Exiting..."; exit ;;
        *) echo "Invalid option" ;;
    esac
done
```

BANKERS:

```
#include <iostream>
using namespace std;
const int max_pro = 10;
const int max_res = 10;
int main() {
int n, m; // Number of processes and resources
cout << "Enter the number of processes: ";
cin >> n;
cout << "Enter the number of resources: ";
cin >> m;
int allocation[max_pro][max_res];
int max[max_pro][max_res];
```

```
int available[max_res];
int need[max_pro][max_res];
// Input allocation matrix
cout << "Enter the allocation matrix: " << endl;
for (int i = 0; i < n; ++i) {
for (int j = 0; j < m; ++j) {
cin >> allocation[i][j];
}
}
// Input max matrix
cout << "Enter the max matrix: " << endl;
for (int i = 0; i < n; ++i) {
for (int j = 0; j < m; ++j) {
cin >> max[i][j];

}
}
// Input available vector
cout << "Enter the available vector: ";
for (int i = 0; i < m; ++i) {
cin >> available[i];
}
// Calculate the need matrix
cout << "Need Matrix:" << endl;
for (int i = 0; i < n; ++i) {
for (int j = 0; j < m; ++j) {
need[i][j] = max[i][j] - allocation[i][j]; cout
<< need[i][j] << " ";
}
cout << endl;
}
bool safe = false;
bool marked[max_pro] = {false}; // Marked processes as safe
int safeSequence[max_pro]; // Store the safe sequence
int safeIndex = 0;
while (!safe) {
safe = true; // Assume all processes are safe
for (int i = 0; i < n; ++i) {
if (!marked[i]) {
bool canExecute = true;
```

```
for (int j = 0; j < m; ++j) {
if (need[i][j] > available[j]) {
canExecute = false;
break;
}
}
if (canExecute) {
marked[i] = true;
safeSequence[safeIndex++] = i; // Add the process to safe sequence for (int j =
0; j < m; ++j) {
available[j] += allocation[i][j]; // Update available resources }
safe = false; // Set safe to false to continue the iteration }
}
}
}
// Print the safe sequence
cout << "Safe Sequence: ";
for (int i = 0; i < safeIndex; ++i) {

cout << "P" << safeSequence[i] << " ";
}
cout << endl;
return 0;
}
```

conditionalllllll

```
#!/bin/bash

# Odd or even
echo "Enter a number:"
read n
if [ $n -eq 0 ]; then
    echo "Neither odd nor even"
elif [ $(expr $n % 2) -eq 0 ]; then
    echo "Even"
```

```
else
    echo "Odd"
fi
echo

# Largest of 3 numbers
echo "Enter 3 numbers:"
read a
read b
read c
if [ $a -ge $b ]; then
    if [ $a -ge $c ]; then
        echo "$a is the greatest number"
    else
        echo "$c is the greatest number"
    fi
else
    if [ $b -ge $c ]; then
        echo "$b is the greatest number"
    else
        echo "$c is the greatest number"
    fi
fi
echo

# Leap year or not
echo "Enter a year:"
read y
if [ $((y % 4)) -eq 0 ] && [ $((y % 100)) -ne 0 ] || [ $((y % 400)) -eq 0 ]; then
    echo "$y is a leap year."
else
    echo "$y is not a leap year."
fi
echo

# Tax calculation
echo "Enter balance: "
read bal
echo "Enter withdrawal: "
read wd
```

```
if (( wd > bal )); then
    echo "Insufficient balance"
else
    tax=0
    if (( wd < 1500 )); then
        tax=$(( wd * 3 / 100 ))
    elif (( wd >= 1500 && wd < 3000 )); then
        tax=$(( wd * 4 / 100 ))
    else
        tax=$(( wd * 5 / 100 ))
    fi
    k=$(( wd - tax ))
    echo "Amount withdrawn: $wd"
    echo "Tax deducted: $tax"
    echo "Amount withdrawn after tax: $k"
fi
```