**1.BANKERS ALGORITHM:**

**#include <stdio.h>**

**int main() {**

**int n, m, i, j, k;**

**printf("Enter the number of processes: ");**

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

**printf("Enter the number of resources: ");**

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

**int alloc[n][m], max[n][m], avail[m];**

**printf("Enter the Allocation Matrix:\n");**

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

**for (j = 0; j < m; j++) {**

**scanf("%d", &alloc[i][j]);**

**}**

**}**

**printf("Enter the MAX Matrix:\n");**

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

**for (j = 0; j < m; j++) {**

**scanf("%d", &max[i][j]);**

**}**

**}**

**printf("Enter the Available Resources:\n");**

**for (j = 0; j < m; j++) {**

**scanf("%d", &avail[j]);**

**}**

**int f[n], ans[n], ind = 0;**

**for (k = 0; k < n; k++) {**

**f[k] = 0;**

**}**

**int need[n][m];**

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

**for (j = 0; j < m; j++)**

**need[i][j] = max[i][j] - alloc[i][j];**

**}**

**int y = 0;**

**for (k = 0; k < n; k++) {**

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

**if (f[i] == 0) {**

**int flag = 0;**

**for (j = 0; j < m; j++) {**

**if (need[i][j] > avail[j]) {**

**flag = 1;**

**break;**

**}**

**}**

**if (flag == 0) {**

**ans[ind++] = i;**

**for (y = 0; y < m; y++)**

**avail[y] += alloc[i][y];**

**f[i] = 1;**

**}**

**}**

**}**

**}**

**int flag = 1;**

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

**if (f[i] == 0) {**

**flag = 0;**

**break;**

**}**

**}**

**if (flag == 0) {**

**printf("The following system is not safe\n");**

**} else {**

**printf("Following is the SAFE Sequence\n");**

**for (i = 0; i < n - 1; i++)**

**printf(" P%d ->", ans[i]);**

**printf(" P%d", ans[n - 1]);**

**}**

**return 0;**

**}**

**2a.FIRST FIT:**

**#include <stdio.h>**

**void firstFit(int blockSize[], int m, int processSize[], int n) {**

**int allocation[n];**

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

**allocation[i] = -1;**

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

**for (int j = 0; j < m; j++) {**

**if (blockSize[j] >= processSize[i]) {**

**allocation[i] = j;**

**blockSize[j] -= processSize[i];**

**break;**

**}**

**}**

**}**

**printf("\nFirst Fit Allocation:\n");**

**printf("Process No.\tProcess Size\tBlock No.\n");**

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

**printf(" %d \t\t %d \t\t", i+1, processSize[i]);**

**if (allocation[i] != -1)**

**printf("%d\n", allocation[i] + 1);**

**else**

**printf("Not Allocated\n");**

**}**

**}**

**int main() {**

**int m, n;**

**printf("Enter the number of memory blocks: ");**

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

**int blockSize[m];**

**printf("Enter the size of each memory block:\n");**

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

**printf("Block %d: ", i+1);**

**scanf("%d", &blockSize[i]);**

**}**

**printf("\nEnter the number of processes: ");**

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

**int processSize[n];**

**printf("Enter the size of each process:\n");**

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

**printf("Process %d: ", i+1);**

**scanf("%d", &processSize[i]);**

**}**

**firstFit(blockSize, m, processSize, n);**

**return 0;**

**}**

**2b.BEST FIT:**

**#include <stdio.h>**

**void bestFit(int blockSize[], int m, int processSize[], int n) {**

**int allocation[n];**

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

**allocation[i] = -1;**

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

**int bestIdx = -1;**

**for (int j = 0; j < m; j++) {**

**if (blockSize[j] >= processSize[i]) {**

**if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])**

**bestIdx = j;**

**}**

**}**

**if (bestIdx != -1) {**

**allocation[i] = bestIdx;**

**blockSize[bestIdx] -= processSize[i];**

**}**

**}**

**printf("\nBest Fit Allocation:\n");**

**printf("Process No.\tProcess Size\tBlock No.\n");**

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

**printf(" %d \t\t %d \t\t", i+1, processSize[i]);**

**if (allocation[i] != -1)**

**printf("%d\n", allocation[i] + 1);**

**else**

**printf("Not Allocated\n");**

**}**

**}**

**int main() {**

**int m, n;**

**printf("Enter the number of memory blocks: ");**

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

**int blockSize[m];**

**printf("Enter the size of each memory block:\n");**

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

**printf("Block %d: ", i+1);**

**scanf("%d", &blockSize[i]);**

**}**

**printf("\nEnter the number of processes: ");**

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

**int processSize[n];**

**printf("Enter the size of each process:\n");**

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

**printf("Process %d: ", i+1);**

**scanf("%d", &processSize[i]);**

**}**

**bestFit(blockSize, m, processSize, n);**

**return 0;**

**}**

**2c.WORST FIT**

**#include <stdio.h>**

**void worstFit(int blockSize[], int m, int processSize[], int n) {**

**int allocation[n];**

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

**allocation[i] = -1;**

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

**int worstIdx = -1;**

**for (int j = 0; j < m; j++) {**

**if (blockSize[j] >= processSize[i]) {**

**if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])**

**worstIdx = j;**

**}**

**}**

**if (worstIdx != -1) {**

**allocation[i] = worstIdx;**

**blockSize[worstIdx] -= processSize[i];**

**}**

**}**

**printf("\nWorst Fit Allocation:\n");**

**printf("Process No.\tProcess Size\tBlock No.\n");**

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

**printf(" %d \t\t %d \t\t", i+1, processSize[i]);**

**if (allocation[i] != -1)**

**printf("%d\n", allocation[i] + 1);**

**else**

**printf("Not Allocated\n");**

**}**

**}**

**int main() {**

**int m, n;**

**printf("Enter the number of memory blocks: ");**

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

**int blockSize[m];**

**printf("Enter the size of each memory block:\n");**

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

**printf("Block %d: ", i+1);**

**scanf("%d", &blockSize[i]);**

**}**

**printf("\nEnter the number of processes: ");**

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

**int processSize[n];**

**printf("Enter the size of each process:\n");**

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

**printf("Process %d: ", i+1);**

**scanf("%d", &processSize[i]);**

**}**

**worstFit(blockSize, m, processSize, n);**

**return 0;**

**}**

**4.combined fits**

**#include <stdio.h>**

**#include <stdlib.h>**

**void firstFit(int blockSize[], int m, int processSize[], int n) {**

**int allocation[n];**

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

**allocation[i] = -1; // Initially no block is assigned to any process**

**}**

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

**for (int j = 0; j < m; j++) {**

**if (blockSize[j] >= processSize[i]) {**

**allocation[i] = j;**

**blockSize[j] -= processSize[i];**

**break;**

**}**

**}**

**}**

**printf("\nProcess No.\tProcess Size\tBlock No.\n");**

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

**printf("%d\t\t%d\t\t", i + 1, processSize[i]);**

**if (allocation[i] != -1)**

**printf("%d\n", allocation[i] + 1);**

**else**

**printf("Not Allocated\n");**

**}**

**}**

**void bestFit(int blockSize[], int m, int processSize[], int n) {**

**int allocation[n];**

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

**allocation[i] = -1; // Initially no block is assigned to any process**

**}**

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

**int bestIdx = -1;**

**for (int j = 0; j < m; j++) {**

**if (blockSize[j] >= processSize[i]) {**

**if (bestIdx == -1 || blockSize[bestIdx] > blockSize[j])**

**bestIdx = j;**

**}**

**}**

**if (bestIdx != -1) {**

**allocation[i] = bestIdx;**

**blockSize[bestIdx] -= processSize[i];**

**}**

**}**

**printf("\nProcess No.\tProcess Size\tBlock No.\n");**

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

**printf("%d\t\t%d\t\t", i + 1, processSize[i]);**

**if (allocation[i] != -1)**

**printf("%d\n", allocation[i] + 1);**

**else**

**printf("Not Allocated\n");**

**}**

**}**

**void worstFit(int blockSize[], int m, int processSize[], int n) {**

**int allocation[n];**

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

**allocation[i] = -1; // Initially no block is assigned to any process**

**}**

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

**int worstIdx = -1;**

**for (int j = 0; j < m; j++) {**

**if (blockSize[j] >= processSize[i]) {**

**if (worstIdx == -1 || blockSize[worstIdx] < blockSize[j])**

**worstIdx = j;**

**}**

**}**

**if (worstIdx != -1) {**

**allocation[i] = worstIdx;**

**blockSize[worstIdx] -= processSize[i];**

**}**

**}**

**printf("\nProcess No.\tProcess Size\tBlock No.\n");**

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

**printf("%d\t\t%d\t\t", i + 1, processSize[i]);**

**if (allocation[i] != -1)**

**printf("%d\n", allocation[i] + 1);**

**else**

**printf("Not Allocated\n");**

**}**

**}**

**int main() {**

**int m, n, choice;**

**printf("Enter the number of blocks: ");**

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

**int blockSize[m];**

**printf("Enter the size of each block:\n");**

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

**printf("Block %d: ", i + 1);**

**scanf("%d", &blockSize[i]);**

**}**

**printf("Enter the number of processes: ");**

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

**int processSize[n];**

**printf("Enter the size of each process:\n");**

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

**printf("Process %d: ", i + 1);**

**scanf("%d", &processSize[i]);**

**}**

**printf("Choose the memory allocation strategy:\n");**

**printf("1. First Fit\n");**

**printf("2. Best Fit\n");**

**printf("3. Worst Fit\n");**

**printf("Enter your choice: ");**

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

**switch (choice) {**

**case 1:**

**firstFit(blockSize, m, processSize, n);**

**break;**

**case 2:**

**bestFit(blockSize, m, processSize, n);**

**break;**

**case 3:**

**worstFit(blockSize, m, processSize, n);**

**break;**

**default:**

**printf("Invalid choice!\n");**

**break;**

**}**

**return 0;**

**}**

**3.Producer consumer problem**

**// C program for the above approach**

**#include <stdio.h>**

**#include <stdlib.h>**

**// Initialize a mutex to 1**

**int mutex = 1;**

**// Number of full slots as 0**

**int full = 0;**

**// Number of empty slots as size**

**// of buffer**

**int empty = 10, x = 0;**

**// Function to produce an item and**

**// add it to the buffer**

**void producer()**

**{**

**// Decrease mutex value by 1**

**--mutex;**

**// Increase the number of full**

**// slots by 1**

**++full;**

**// Decrease the number of empty**

**// slots by 1**

**--empty;**

**// Item produced**

**x++;**

**printf("\nProducer produces"**

**"item %d",**

**x);**

**// Increase mutex value by 1**

**++mutex;**

**}**

**// Function to consume an item and**

**// remove it from buffer**

**void consumer()**

**{**

**// Decrease mutex value by 1**

**--mutex;**

**// Decrease the number of full**

**// slots by 1**

**--full;**

**// Increase the number of empty**

**// slots by 1**

**++empty;**

**printf("\nConsumer consumes "**

**"item %d",**

**x);**

**x--;**

**// Increase mutex value by 1**

**++mutex;**

**}**

**// Driver Code**

**int main()**

**{**

**int n, i;**

**printf("\n1. Press 1 for Producer"**

**"\n2. Press 2 for Consumer"**

**"\n3. Press 3 for Exit");**

**// Using '#pragma omp parallel for'**

**// can give wrong value due to**

**// synchronization issues.**

**// 'critical' specifies that code is**

**// executed by only one thread at a**

**// time i.e., only one thread enters**

**// the critical section at a given time**

**#pragma omp critical**

**for (i = 1; i > 0; i++) {**

**printf("\nEnter your choice:");**

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

**// Switch Cases**

**switch (n) {**

**case 1:**

**// If mutex is 1 and empty**

**// is non-zero, then it is**

**// possible to produce**

**if ((mutex == 1)**

**&& (empty != 0)) {**

**producer();**

**}**

**// Otherwise, print buffer**

**// is full**

**else {**

**printf("Buffer is full!");**

**}**

**break;**

**case 2:**

**// If mutex is 1 and full**

**// is non-zero, then it is**

**// possible to consume**

**if ((mutex == 1)**

**&& (full != 0)) {**

**consumer();**

**}**

**// Otherwise, print Buffer**

**// is empty**

**else {**

**printf("Buffer is empty!");**

**}**

**break;**

**// Exit Condition**

**case 3:**

**exit(0);**

**break;**

**}**

**}**

**}**

**4.Page table:**

**#include<stdio.h>**

**#include<stdlib.h>**

**int main() {**

**int n = 10;**

**int arr[10];**

**int p;**

**int d;**

**int i;**

**int physicaladd;**

**// Accepting dynamic input for the array**

**printf("Enter 10 values for the array:\n");**

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

**printf("Enter value for arr[%d]: ", i);**

**scanf("%d", &arr[i]);**

**}**

**while(1) {**

**printf("Enter 1 for PageNo and Displacement \nEnter 2 to exit program \n");**

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

**switch(i) {**

**case 1:**

**printf("Enter pageno: ");**

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

**if(p < 0 || p >= n) {**

**printf("Invalid pageno. Please enter a value between 0 and 9.\n");**

**break;**

**}**

**printf("Enter displacement: ");**

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

**physicaladd = arr[p] + d;**

**printf("The physical address is %d \n", physicaladd);**

**break;**

**case 2:**

**printf("Exiting the program.\n");**

**exit(0);**

**default:**

**printf("Invalid choice. Please enter 1 or 2.\n");**

**}**

**}**

**return 0;**

**}**

**5.FCFS:**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct Process {**

**int id;**

**int arrival\_time;**

**int burst\_time;**

**int waiting\_time;**

**int turnaround\_time;**

**};**

**int compareProcesses(const void\* a, const void\* b) {**

**struct Process\* process1 = (struct Process\*)a;**

**struct Process\* process2 = (struct Process\*)b;**

**if (process1->arrival\_time != process2->arrival\_time)**

**return process1->arrival\_time - process2->arrival\_time;**

**else**

**return process1->id - process2->id;**

**}**

**// Function to calculate waiting time, turn around time, and draw Gantt chart**

**void calculateAndDraw(int n, struct Process processes[]) {**

**// Sort the processes based on arrival time and process ID**

**qsort(processes, n, sizeof(struct Process), compareProcesses);**

**// Calculate waiting time and turn around time**

**int completion\_time[n], total\_wt = 0, total\_tat = 0;**

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

**if (i == 0)**

**completion\_time[i] = processes[i].burst\_time;**

**else**

**completion\_time[i] = completion\_time[i - 1] + processes[i].burst\_time;**

**// Calculate waiting time**

**processes[i].waiting\_time = completion\_time[i] - processes[i].burst\_time - processes[i].arrival\_time;**

**if (processes[i].waiting\_time < 0)**

**processes[i].waiting\_time = 0;**

**total\_wt += processes[i].waiting\_time;**

**// Calculate turn around time**

**processes[i].turnaround\_time = completion\_time[i] - processes[i].arrival\_time;**

**total\_tat += processes[i].turnaround\_time;**

**}**

**// Print Gantt chart**

**printf("\nGantt Chart:\n");**

**printf(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");**

**printf("|");**

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

**printf(" P%d |", processes[i].id);**

**}**

**printf("\n");**

**printf("|");**

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

**printf(" %d |", completion\_time[i]);**

**}**

**printf("\n");**

**// Print WT and TAT for each process**

**printf("\nProcess Burst time Arrival time Waiting time Turnaround time\n");**

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

**printf(" %d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burst\_time, processes[i].arrival\_time, processes[i].waiting\_time, processes[i].turnaround\_time);**

**}**

**// Print average waiting time and turn around time**

**float avg\_wt = (float)total\_wt / n;**

**float avg\_tat = (float)total\_tat / n;**

**printf("\nAverage Waiting Time: %.2f\n", avg\_wt);**

**printf("Average Turnaround Time: %.2f\n", avg\_tat);**

**}**

**int main() {**

**int n;**

**printf("Enter the number of processes: ");**

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

**struct Process processes[n];**

**printf("Enter burst time and arrival time for each process:\n");**

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

**printf("Process %d:\n", i + 1);**

**printf("Burst time: ");**

**scanf("%d", &processes[i].burst\_time);**

**printf("Arrival time: ");**

**scanf("%d", &processes[i].arrival\_time);**

**processes[i].id = i + 1;**

**}**

**calculateAndDraw(n, processes);**

**return 0;**

**}**

**6.SJF:**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include <limits.h>**

**struct Process {**

**int id;**

**int arrival\_time;**

**int burst\_time;**

**int waiting\_time;**

**int turnaround\_time;**

**};**

**int compareProcesses(const void\* a, const void\* b) {**

**struct Process\* process1 = (struct Process\*)a;**

**struct Process\* process2 = (struct Process\*)b;**

**if (process1->arrival\_time != process2->arrival\_time)**

**return process1->arrival\_time - process2->arrival\_time;**

**else**

**return process1->burst\_time - process2->burst\_time;**

**}**

**void calculateAndDraw(int n, struct Process processes[]) {**

**qsort(processes, n, sizeof(struct Process), compareProcesses);**

**int remaining\_time[n];**

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

**remaining\_time[i] = processes[i].burst\_time;**

**}**

**int completion\_time[n];**

**int time = 0;**

**while (1) {**

**int shortest\_burst\_index = -1;**

**int shortest\_burst = INT\_MAX;**

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

**if (processes[i].arrival\_time <= time && remaining\_time[i] < shortest\_burst && remaining\_time[i] > 0) {**

**shortest\_burst = remaining\_time[i];**

**shortest\_burst\_index = i;**

**}**

**}**

**if (shortest\_burst\_index == -1)**

**break;**

**time += remaining\_time[shortest\_burst\_index];**

**completion\_time[shortest\_burst\_index] = time;**

**remaining\_time[shortest\_burst\_index] = 0;**

**processes[shortest\_burst\_index].waiting\_time = time - processes[shortest\_burst\_index].arrival\_time - processes[shortest\_burst\_index].burst\_time;**

**if (processes[shortest\_burst\_index].waiting\_time < 0)**

**processes[shortest\_burst\_index].waiting\_time = 0;**

**processes[shortest\_burst\_index].turnaround\_time = time - processes[shortest\_burst\_index].arrival\_time;**

**}**

**int total\_wt = 0, total\_tat = 0;**

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

**total\_wt += processes[i].waiting\_time;**

**total\_tat += processes[i].turnaround\_time;**

**}**

**printf("\nGantt Chart:\n");**

**printf(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");**

**printf("|");**

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

**printf(" P%d |", processes[i].id);**

**}**

**printf("\n");**

**printf("|");**

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

**printf(" %d |", completion\_time[i]);**

**}**

**printf("\n");**

**printf("\nProcess Burst time Arrival time Waiting time Turnaround time\n");**

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

**printf(" %d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burst\_time, processes[i].arrival\_time, processes[i].waiting\_time, processes[i].turnaround\_time);**

**}**

**float avg\_wt = (float)total\_wt / n;**

**float avg\_tat = (float)total\_tat / n;**

**printf("\nAverage Waiting Time: %.2f\n", avg\_wt);**

**printf("Average Turnaround Time: %.2f\n", avg\_tat);**

**}**

**int main() {**

**int n;**

**printf("Enter the number of processes: ");**

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

**struct Process processes[n];**

**printf("Enter burst time and arrival time for each process:\n");**

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

**printf("Process %d:\n", i + 1);**

**printf("Burst time: ");**

**scanf("%d", &processes[i].burst\_time);**

**printf("Arrival time: ");**

**scanf("%d", &processes[i].arrival\_time);**

**processes[i].id = i + 1;**

**}**

**calculateAndDraw(n, processes);**

**return 0;}**

**7.NON PREEMPTIVE:**

**#include <stdio.h>**

**int main() {**

**int n; // Number of Processes**

**printf("Enter the number of processes: ");**

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

**int arrivaltime[n], bursttime[n], priority[n], waitingTime[n], turnaroundTime[n];**

**int CPU = 0, allTime = 0;**

**printf("Enter arrival time, burst time, and priority for each process:\n");**

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

**printf("For Process %d:\n", i + 1);**

**printf("Arrival Time: ");**

**scanf("%d", &arrivaltime[i]);**

**printf("Burst Time: ");**

**scanf("%d", &bursttime[i]);**

**printf("Priority: ");**

**scanf("%d", &priority[i]);**

**}**

**int ATt[n], PPt[n];**

**int NoP = n;**

**int i = 0;**

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

**PPt[i] = priority[i];**

**ATt[i] = arrivaltime[i];**

**}**

**int LAT = 0;**

**for (i = 0; i < n; i++)**

**if (arrivaltime[i] > LAT)**

**LAT = arrivaltime[i];**

**int MAX\_P = 0;**

**for (i = 0; i < n; i++)**

**if (PPt[i] > MAX\_P)**

**MAX\_P = PPt[i];**

**int ATi = 0, P1 = PPt[0], P2 = PPt[0];**

**int j = -1;**

**while (NoP > 0 && CPU <= 1000) {**

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

**if ((ATt[i] <= CPU) && (ATt[i] != (LAT + 10))) {**

**if (PPt[i] != (MAX\_P + 1)) {**

**P2 = PPt[i];**

**j = 1;**

**if (P2 < P1) {**

**j = 1;**

**ATi = i;**

**P1 = PPt[i];**

**P2 = PPt[i];**

**}**

**}**

**}**

**}**

**if (j == -1) {**

**CPU = CPU + 1;**

**continue;**

**} else {**

**waitingTime[ATi] = CPU - ATt[ATi];**

**CPU = CPU + bursttime[ATi];**

**turnaroundTime[ATi] = CPU - ATt[ATi];**

**ATt[ATi] = LAT + 10;**

**j = -1;**

**PPt[ATi] = MAX\_P + 1;**

**ATi = 0;**

**P1 = MAX\_P + 1;**

**P2 = MAX\_P + 1;**

**NoP = NoP - 1;**

**}**

**}**

**printf("\nProcess\_Number\tBurst\_Time\tPriority\tArrival\_Time\tWaiting\_Time\tTurnaround\_Time\n\n");**

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

**printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, bursttime[i], priority[i], arrivaltime[i], waitingTime[i], turnaroundTime[i]);**

**}**

**float AvgWT = 0, AVGTaT = 0;**

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

**AvgWT = waitingTime[i] + AvgWT;**

**AVGTaT = turnaroundTime[i] + AVGTaT;**

**}**

**printf("Average waiting time = %f\n", AvgWT / n);**

**printf("Average turnaround time = %f\n", AVGTaT / n);**

**return 0;**

**}**

**Shell program:**

**Largest of three number**

echo "Enter Num1"

read num1

echo "Enter Num2"

read num2

echo "Enter Num3"

read num3

if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]

then

echo $num1

elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]

then

echo $num2

else

echo $num3

fi**factorial number**

echo "Enter a number"

read num

fact=1

**while** [ $num -gt 1 ]

**do**

fact=**$((**fact \* num**))** #fact = fact \* num

num=**$((**num - **1))** #num = num - 1

**done**

echo $fact

**sum of digits:**

**#!/bin/bash**

echo "Enter a Number:"

read n

**temp**=**$n**

**sd**=0

**sum**=0

**while** [ **$n** -**gt** 0 ]

**do**

**sd**=$(( **$n** % 10 ))

**n**=$(( **$n** / 10 ))

**sum**=$(( **$sum** + **$sd** ))

**done**

echo "Sum is $sum"

**reverse a number:**

echo enter n

read n

**num**=0

**while** [ **$n** -**gt** 0 ]

**do**

**num**=$(expr **$num** \\* 10)

**k**=$(expr **$n** % 10)

**num**=$(expr **$num** + **$k**)

**n**=$(expr **$n** / 10)

**done**

echo number is **$num**

**Fibonacci series:**

**echo -n "Enter the number of terms: "**

**read n**

**# Initialize variables**

**a=0**

**b=1**

**# Print the first two terms**

**echo -n "$a $b "**

**# Generate the Fibonacci series**

**for (( i=2; i<$n; i++ ))**

**do**

**c=$((a + b))**

**echo -n "$c "**

**a=$b**

**b=$c**

**done**

**echoArmstrong number:**

**echo "Enter A Number: "**

**read n**

**arm=0**

**temp=$n**

**while [ "$n" -ne 0 ]; do**

**r=$((n % 10))**

**arm=$((arm + r \* r \* r))**

**n=$((n / 10))**

**done**

**if [ $arm -eq "$temp" ]; then**

**echo "Armstrong"**

**else**

**echo "Not Armstrong"**

**fi**