Exercise no : 2(a) FCFS : [fcfs.c]

```
#include <stdio.h>

int main() {
	int i, n;
	printf("Enter the number of processes: ");
	scanf("%d", &n);

	int process[n], bursttime[n], waittime[n], turnaround[n];
	float avgwaittime = 0, avgturnaroundtime = 0;

	printf("Enter the burst time for each process:\n");
	for (i = 0; i < n; i++) {
	process[i] = i + 1;
	printf("Process %d: ", i + 1);
	scanf("%d", &bursttime[i]);
	}

	waittime[0] = 0;
	for (i = 1; i < n; i++) {
	waittime[i] = waittime[i - 1] + bursttime[i - 1];
	}

	for (i = 0; i < n; i++) {
	turnaround[i] = waittime[i] + bursttime[i];
	}

	for (i = 0; i < n; i++) {
	avgwaittime += waittime[i];
	avgturnaroundtime += turnaround[i];
	}
	avgwaittime /= n;
	avgturnaroundtime /= n;

	printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround
Time\n");
	for (i = 0; i < n; i++) {
```

```
       printf("P%d\t%d\t\t%d\t\t%d\n", process[i], bursttime[i],
waittime[i], turnaround[i]);
       }

       printf("Average Waiting Time: %.2f\n", avgwaittime);
       printf("Average Turnaround Time: %.2f\n",
avgturnaroundtime);

       printf("\nGantt Chart:\n");
       printf(" +");
       for (i = 0; i < n; i++) {
       printf("----+");
       }
       printf("\n");

       printf(" |");
       for (i = 0; i < n; i++) {
       printf(" P%d |", process[i]);
       }
       printf("\n");

       printf(" +");
       for (i = 0; i < n; i++) {
       printf("----+");
       }
       printf("\n");

       printf(" 0");
       for (i = 0; i < n; i++) {
       printf("   %d", turnaround[i]);
       }
       printf("\n");
       return 0;
}
```

```
Enter the number of processes: 5
Enter the burst time for each process:
Process 1: 10
Process 2: 1
Process 3: 2
Process 4: 1
Process 5: 5

Process Burst Time      Waiting Time    Turnaround Time
P1      10              0               10
P2      1               10              11
P3      2               11              13
P4      1               13              14
P5      5               14              19
Average Waiting Time: 9.60
Average Turnaround Time: 13.40

Gantt Chart:
 +----+----+----+----+----+
 | P1 | P2 | P3 | P4 | P5 |
 +----+----+----+----+----+
 0   10   11   13   14   19
```

Exercise no : 2(b) SJF : [sjf.c]

```
#include<stdio.h>
void sortByBurstTime(int n, int process[], int bursttime[]) {
	int temp, i, j;
	for(i = 0; i < n-1; i++) {
	for(j = i+1; j < n; j++) {
	if(bursttime[i] > bursttime[j]) {
		temp = bursttime[i];
		bursttime[i] = bursttime[j];
		bursttime[j] = temp;
		temp = process[i];
		process[i] = process[j];
		process[j] = temp;
	}
	}
	}
}

int main(){
	int i, n;
	printf("Enter the number of processes: ");
	scanf("%d", &n);
```

```
        int process[n], bursttime[n], waittime[n], turnaround[n];
        float avgwaittime = 0, avgturnaroundtime = 0;
        printf("Enter the burst time for each process:\n");
        for(i = 0; i < n; i++) {
        process[i] = i + 1;
        printf("Process %d: ", i + 1);
        scanf("%d", &bursttime[i]);
        }
        sortByBurstTime(n, process, bursttime);
        waittime[0] = 0;
        for(i = 1; i < n; i++) {
        waittime[i] = waittime[i-1] + bursttime[i-1];
        }
        for(i = 0; i < n; i++) {
        turnaround[i] = waittime[i] + bursttime[i];
        }
        for(i = 0; i < n; i++) {
        avgwaittime += waittime[i];
        avgturnaroundtime += turnaround[i];
        }
        avgwaittime /= n;
        avgturnaroundtime /= n;
        printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround
Time\n");
        for(i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\t\t%d\n", process[i], bursttime[i],
waittime[i], turnaround[i]);
        }
        printf("Average Waiting Time: %.2f\n", avgwaittime);
        printf("Average Turnaround Time: %.2f\n",
avgturnaroundtime);
        printf("\nGantt Chart:\n");
        printf(" +");
        for(i = 0; i < n; i++) {
        printf("----+");
        }
        printf("\n");
        printf(" |");
        for(i = 0; i < n; i++) {
```

```
        printf(" P%d |", process[i]);
        }
        printf("\n");
        printf(" +");
        for(i = 0; i < n; i++) {
        printf("----+");
        }
        printf("\n");
        printf(" 0");
        for(i = 0; i < n; i++) {
        printf("   %d", turnaround[i]);
        }
        printf("\n");

        return 0;
}]
```

```
Enter the number of processes: 4
Enter the burst time for each process:
Process 1: 5
Process 2: 2
Process 3: 3
Process 4: 1

Process Burst Time      Waiting Time    Turnaround Time
P4      1               0               1
P2      2               1               3
P3      3               3               6
P1      5               6               11
Average Waiting Time: 2.50
Average Turnaround Time: 5.25

Gantt Chart:
 +----+----+----+----+
 | P4 | P2 | P3 | P1 |
 +----+----+----+----+
 0   1   3   6   11
```

---

https://www.log2base2.com/C/basic/how-to-compile-the-c-program.html

Type your program.

4.To save the file:

Press Esc button and then type :wq. It will save the file.

5.gcc file.c   // To compile the program

6. ./a.out   // To Run the program

VIVA QUESTIONS :

https://www.gatevidyalay.com/round-robin-round-robin-scheduling-examples/

—---

Exercise no : 3(a) Priority Scheduling Algorithm :

1. Input number of processes N.

2. Input burst time and priority for each process.

3. Sort the processes based on priority (in descending order).

4. Initialize variables for waiting time and turnaround time for each process.

5. Compute waiting time for each process:

   - waiting_time[i] = sum of burst times of all previous processes

6. Compute turnaround time for each process:

   - turnaround_time[i] = burst_time[i] + waiting_time[i]

7. Output waiting time and turnaround time for each process.

8. Calculate average waiting time and average turnaround time.

Exercise no : 3(a) Priority Scheduling Algorithm : [psa.c]

```
#include <stdio.h>
#include <stdlib.h>

void sortPriority(int n, int burstTime[], int priority[], int process[]) {
    int i, j;
```

```
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (priority[j] > priority[j + 1]) {
                int temp = priority[j];
                priority[j] = priority[j + 1];
                priority[j + 1] = temp;

                temp = burstTime[j];
                burstTime[j] = burstTime[j + 1];
                burstTime[j + 1] = temp;

                temp = process[j];
                process[j] = process[j + 1];
                process[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n, i;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    int *burstTime = (int*)malloc(n * sizeof(int));
    int *priority = (int*)malloc(n * sizeof(int));
    int *process = (int*)malloc(n * sizeof(int));
    int *waitingTime = (int*)malloc(n * sizeof(int));
    int *turnaroundTime = (int*)malloc(n * sizeof(int));
```

```
printf("Enter the burst time and priority for each process:\n");
for (i = 0; i < n; i++) {
    process[i] = i + 1;
    printf("Process %d:\n", i + 1);
    printf("Burst Time: ");
    scanf("%d", &burstTime[i]);
    printf("Priority: ");
    scanf("%d", &priority[i]);
}

sortPriority(n, burstTime, priority, process);

waitingTime[0] = 0;
for (i = 1; i < n; i++) {
    waitingTime[i] = waitingTime[i - 1] + burstTime[i - 1];
}

for (i = 0; i < n; i++) {
    turnaroundTime[i] = waitingTime[i] + burstTime[i];
}

float avgWaitingTime = 0, avgTurnaroundTime = 0;
for (i = 0; i < n; i++) {
    avgWaitingTime += waitingTime[i];
    avgTurnaroundTime += turnaroundTime[i];
}
avgWaitingTime /= n;
avgTurnaroundTime /= n;
```

```
    printf("\nGantt Chart:\n");
    printf(" -------------------------------\n");
    printf(" | ");
    for (i = 0; i < n; i++) {
        printf(" P%d | ", process[i]);
    }
    printf("\n -------------------------------\n");
    printf("0");
    int currentTime = 0;
    for (i = 0; i < n; i++) {
        currentTime += burstTime[i];
        printf("\t %d", currentTime);
    }
    printf("\n\n");

    printf("Process\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("P%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t\t%d\n", process[i], burstTime[i],
priority[i], waitingTime[i], turnaroundTime[i]);
    }

    printf("\nAverage Waiting Time: %.2f ms\n", avgWaitingTime);
    printf("Average Turnaround Time: %.2f ms\n", avgTurnaroundTime);

    free(burstTime);
    free(priority);
    free(process);
    free(waitingTime);
```

```c
    free(turnaroundTime);


    return 0;
}
```

```
Enter the number of processes: 5
Process 1 Burst Time: 10
Priority: 3
Process 2 Burst Time: 1
Priority: 1
Process 3 Burst Time: 2
Priority: 3
Process 4 Burst Time: 1
Priority: 4
Process 5 Burst Time: 5
Priority: 2

Gantt Chart:
+-----+-----+-----+-----+-----+
|  P2 |  P5 |  P1 |  P3 |  P4 |
+-----+-----+-----+-----+-----+
0    1    6    16    18    19

Process Burst Time      Priority        Waiting Time    Turnaround Time
P2              1               1               0               1
P5              5               2               1               6
P1              10              3               6               16
P3              2               3               16              18
P4              1               4               18              19

Average Waiting Time: 8.20 ms
Average Turnaround Time: 12.00 ms
```

Exercise no : 3(b) ROUND ROBIN :

```c
#include <stdio.h>
struct Process {
        int id, bt, wt, tat, remaining_bt;
};
void printGanttChart (int processOrder[], int timeStamps[], int count) {
    printf("\n Gantt Chart:\n");
        printf(" ");
          for (int i = 0; i < count; i++)
                   printf("-----");
    printf("\n|");

        for (int i = 0; i < count; i++)
                printf(" P%d |", processOrder[i]);
    printf("\n ");
```

```c
    for (int i = 0; i < count; i++)
        printf("-----");
    printf("\n");

    for (int i = 0; i < count; i++)
        printf("   %2d", timeStamps[i]);
    printf("\n");
}

void roundRobinScheduling (int n, struct Process processes[], int quantum) {
    int time = 0, completed = 0;
    int processOrder[100], timeStamps[100], count = 0;

    while (completed < n) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (processes[i].remaining_bt > 0) {
                done = 0;
                processOrder[count] = processes[i].id;
                if (processes[i].remaining_bt > quantum) {
                    time += quantum;
                    processes[i].remaining_bt -= quantum;
                } else {
                    time += processes[i].remaining_bt;
                    processes[i].wt = time - processes[i].bt;
                    processes[i].remaining_bt = 0;
                    completed++;
                }
                timeStamps[count++] = time;
            }
        }
        if (done) break;
    }
    for (int i = 0; i < n; i++) {
        processes[i].tat = processes[i].bt + processes[i].wt;
    }

    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++)
```

```
        printf("P%d\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].bt,
processes[i].wt, processes[i].tat);
        printGanttChart(processOrder, timeStamps, count);
        float totalWT = 0, totalTAT = 0;
        for (int i = 0; i < n; i++) {
        totalWT += processes[i].wt;
        totalTAT += processes[i].tat;
        }
        printf("\nAverage Waiting Time: %.2f ms", totalWT / n);
        printf("\nAverage Turnaround Time: %.2f ms\n", totalTAT / n);
}

int main() {
        int n, quantum;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
        struct Process processes[n];
        for (int i = 0; i < n; i++) {
                processes[i].id = i + 1;
                printf("Enter burst time for process P%d: ", i + 1);
                scanf("%d", &processes[i].bt);
                processes[i].remaining_bt = processes[i].bt;
        }
         printf("Enter time quantum: ");
         scanf("%d", &quantum);
         roundRobinScheduling(n, processes, quantum);
        return 0;
}
```

```
Enter the number of processes: 3
Enter burst time for process P1: 24
Enter burst time for process P2: 3
Enter burst time for process P3: 3
Enter time quantum: 4

Process Burst Time      Waiting Time    Turnaround Time
P1      24              6               30
P2      3               4               7
P3      3               7               10

Gantt Chart:
 ----------------------------------------
| P1 | P2 | P3 | P1 | P1 | P1 | P1 | P1 |
 ----------------------------------------
0    4    7   10   14   18   22   26   30

Average Waiting Time: 5.67 ms
Average Turnaround Time: 15.67 ms
```

Experiment 4(a) : Interprocess Communication using Shared Memory

Server.c :

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
int main() {
        key_t key = ftok("shmfile", 65);  // Generate a unique key
        int shmid = shmget(key, 1024, 0666 | IPC_CREAT); // Create shared
memory segment
        char *str = (char *)shmat(shmid, (void *)0, 0); // Attach shared memory
           printf("Write Data: ");
        fgets(str, 1024, stdin);  // Get input from the user
           printf("Data written to shared memory: %s\n", str);
        shmdt(str);
        return 0;
}
```

Client.c :

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int main() {
        key_t key = ftok("shmfile", 65); // Generate the same key
        int shmid = shmget(key, 1024, 0666 | IPC_CREAT); // Get shared memory
        char *str = (char *)shmat(shmid, (void *)0, 0); // Attach shared memory
          printf("Data read from shared memory: %s\n", str);
        shmdt(str); // Detach shared memory
        shmctl(shmid, IPC_RMID, NULL); // Destroy shared memory
        return 0;
}
```

OUTPUT :

```
~/2025-02-06-file-1/2025-02-06-file-1/2025-02-06-file-1$ vi server.c
~/2025-02-06-file-1/2025-02-06-file-1/2025-02-06-file-1$ vi client2.c
~/2025-02-06-file-1/2025-02-06-file-1/2025-02-06-file-1$ gcc server.c -o server
~/2025-02-06-file-1/2025-02-06-file-1/2025-02-06-file-1$ ./server
Write Data: Bala is CSE student
Data written to shared memory: Bala is CSE student

~/2025-02-06-file-1/2025-02-06-file-1/2025-02-06-file-1$ gcc client2.c -o client2
~/2025-02-06-file-1/2025-02-06-file-1/2025-02-06-file-1$ ./client2
Data read from shared memory: Bala is CSE student

~/2025-02-06-file-1/2025-02-06-file-1/2025-02-06-file-1$
```

—---

Experiment 4(b) : Interprocess Communication using Pipes

Option 1 :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdbool.h>

#define N 25

bool is_prime(int n) {
    if (n < 2) return false;
    for (int i = 2; i * i <= n; i++)
```

```c
        if (n % i == 0) return false;
    return true;
}

int main() {
    int fd[2], fib[N];
    pipe(fd);
    if (fork() == 0) { // Child
        close(fd[1]);
        read(fd[0], fib, sizeof(fib));
        close(fd[0]);
        printf("Child (Primes in Fibonacci):\n");
        for (int i = 0; i < N; i++)
            if (is_prime(fib[i])){
                        printf("%d ", fib[i]);
            }
        printf("\n");
    } else { // Parent
        close(fd[0]);
        fib[0] = 0; fib[1] = 1;
        for (int i = 2; i < N; i++)
            fib[i] = fib[i - 1] + fib[i - 2];
        write(fd[1], fib, sizeof(fib));
        close(fd[1]);
        printf("Parent (Fibonacci Series):\n");
        for (int i = 0; i < N; i++)
            printf("%d ", fib[i]);
        printf("\n");
    }
    return 0;
}
```

```
Parent (Fibonacci Series):
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946
    17711 28657 46368
Child (Primes in Fibonacci):
2 3 5 13 89 233 1597 28657
```

Option 2 :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void generate_fibonacci(int n, int fd[]) {
    close(fd[0]);

    int a = 0, b = 1, next;
    write(fd[1], &a, sizeof(int));
    if (n > 1)
        write(fd[1], &b, sizeof(int));

    for (int i = 2; i < n; i++) {
        next = a + b;
        write(fd[1], &next, sizeof(int));
        a = b;
        b = next;
    }
    close(fd[1]);
}

void read_fibonacci(int fd[]) {
    close(fd[1]);

    int num;
    printf("Fibonacci Sequence: ");
    while (read(fd[0], &num, sizeof(int)) > 0) {
        printf("%d ", num);
    }
    printf("\n");
    close(fd[0]);
}

int main() {
    int fd[2];
    pid_t pid;
    int n;

    printf("Enter number of Fibonacci terms: ");
    scanf("%d", &n);
```

```
    if (pipe(fd) == -1) {
        perror("Pipe failed");
        exit(1);
    }

    pid = fork();
    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    }

    if (pid > 0) {
        generate_fibonacci(n, fd);
    } else {
        read_fibonacci(fd);
    }

    return 0;
}
```

OUTPUT :

```
Output                                                    Clear
Enter number of Fibonacci terms: 10
Fibonacci Sequence: 0 1 1 2 3 5 8 13 21 34


=== Code Execution Successful ===
```

—---

Experiment 4(c) : InterProcess Communication using Message Queue

Server.c :

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```c
#include <string.h>

struct msg_buffer {
	long msg_type;
	char msg_text[100];
} message;

int main() {
	int msgid;
	key_t key = ftok("progfile", 65);
	msgid = msgget(key, 0666 | IPC_CREAT);
	msgrcv(msgid, &message, sizeof(message), 1, 0);
	printf("Server received: %s\n", message.msg_text);
	msgctl(msgid, IPC_RMID, NULL);
	return 0;
}
```

Client.c :

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>

struct msg_buffer {
	long msg_type;
	char msg_text[100];
} message;

int main() {
	int msgid;
	key_t key = ftok("progfile", 65);
	msgid = msgget(key, 0666 | IPC_CREAT);
	message.msg_type = 1;
	printf("Enter message: ");
	fgets(message.msg_text, sizeof(message.msg_text), stdin);
	msgsnd(msgid, &message, sizeof(message), 0);
```

```
        printf("Client sent: %s", message.msg_text);
        return 0;
}
```

OUTPUT :

```
~$ vi client.c
~$ gcc client.c
~$ ./a.out
Enter message: Hello Swetha!!
Client sent: Hello Swetha!!
```

```
~$ vi server24.c
~$ gcc server24.c
~$ ./a.out
Server received: Hello Swetha!!
```

—---

Exercise 5 : Producer Consumer Problem using Semaphore

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define BUFFER_SIZE 10
#define MAX_ITEMS 20

int buffer[BUFFER_SIZE];
int in = 0, out = 0;
sem_t empty, full, mutex;
int produced_items = 0, consumed_items = 0;

void *producer(void *param) {
    int item;
    while (produced_items < MAX_ITEMS) {
        item = rand() % 100;
        sem_wait(&empty);
        sem_wait(&mutex);
        buffer[in] = item;
        printf("Producer produces item %d\n", item);
        in = (in + 1) % BUFFER_SIZE;
        sem_post(&mutex);
        sem_post(&full);
```

```
        produced_items++;
        sleep(rand() % 2);
    }
    printf("Producer has produced %d items.\n", produced_items);
    return NULL;
}
void *consumer(void *param) {
    int item;
    while (consumed_items < MAX_ITEMS) {
        sem_wait(&full);
        sem_wait(&mutex);
        item = buffer[out];
        printf("Consumer consumes item %d\n", item);
        out = (out + 1) % BUFFER_SIZE;
        sem_post(&mutex);
        sem_post(&empty);
        consumed_items++;
        sleep(rand() % 2);
    }
    printf("Consumer has consumed %d items.\n", consumed_items);
    return NULL;
}


int main() {
    pthread_t producer_thread, consumer_thread;
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    sem_init(&mutex, 0, 1);
    pthread_create(&producer_thread, NULL, producer, NULL);
    pthread_create(&consumer_thread, NULL, consumer, NULL);
    pthread_join(producer_thread, NULL);
    pthread_join(consumer_thread, NULL);
    sem_destroy(&empty);
    sem_destroy(&full);
    sem_destroy(&mutex);
    return 0;
}
```

Producer produced: 83

OUTPUT :

Consumer consumed: 83
Producer produced: 86
Consumer consumed: 86
Producer produced: 77
Consumer consumed: 77
Producer produced: 15
Consumer consumed: 15
Producer produced: 93
Consumer consumed: 93
Producer produced: 35
Consumer consumed: 35
Producer produced: 86
Consumer consumed: 86

Exercise no : 6 - Bankers Algorithm

```
#include <stdio.h>
#include <stdbool.h>

int main() {
    int P, R;

    printf("Enter the number of processes: ");
    scanf("%d", &P);
    printf("Enter the number of resources: ");
    scanf("%d", &R);

    int alloc[P][R], max[P][R], need[P][R], avail[R], work[R];
    bool finish[P];
    int safeSeq[P];

    // Input Allocation Matrix
    printf("Enter the Allocation Matrix:\n");
    for (int i = 0; i < P; i++)
```

```
    for (int j = 0; j < R; j++)
        scanf("%d", &alloc[i][j]);

// Input Max Matrix
printf("Enter the Max Matrix:\n");
for (int i = 0; i < P; i++)
    for (int j = 0; j < R; j++)
        scanf("%d", &max[i][j]);

// Input Available Resources
printf("Enter the Available Resources:\n");
for (int i = 0; i < R; i++)
    scanf("%d", &avail[i]);

// Calculate Need matrix
for (int i = 0; i < P; i++)
    for (int j = 0; j < R; j++)
        need[i][j] = max[i][j] - alloc[i][j];

// Copy avail to work and initialize finish[]
for (int i = 0; i < R; i++)
    work[i] = avail[i];
for (int i = 0; i < P; i++)
    finish[i] = false;

int count = 0;

while (count < P) {
    bool found = false;
    for (int p = 0; p < P; p++) {
        if (!finish[p]) {
            int j;
            for (j = 0; j < R; j++)
                if (need[p][j] > work[j])
                    break;

            if (j == R) {
                for (int k = 0; k < R; k++)
```

```
                work[k] += alloc[p][k];

            safeSeq[count++] = p;
            finish[p] = true;
            found = true;
          }
        }
      }

      if (!found) {
        printf("\nSystem is not in a safe state (deadlock possible).\n");
        return -1;
      }
    }

    // Safe state output
    printf("\nSystem is in a safe state.\nSafe Sequence:\n");

    // Reset work to initial available
    for (int i = 0; i < R; i++)
      work[i] = avail[i];

    for (int i = 0; i < P; i++) {
      int p = safeSeq[i];
      printf("P%d | Available: ", p);
      for (int j = 0; j < R; j++)
        printf("R%d: %d ", j + 1, work[j]);

      // Update available (work) by adding this process's allocation
      for (int j = 0; j < R; j++)
        work[j] += alloc[p][j];
      printf("\n---------------------------------------------------------\n");
    }

    return 0;
}
```

```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the Max Matrix:
7 5 3
3 3 2
9 0 2
2 2 2
4 3 3
Enter the Available Resources:
3 3 2
```

```
System is in a safe state.
Safe Sequence:
P1 | Available: R1: 3 R2: 3 R3: 2
----------------------------------------------------------
P3 | Available: R1: 5 R2: 3 R3: 2
----------------------------------------------------------
P4 | Available: R1: 7 R2: 4 R3: 3
----------------------------------------------------------
P0 | Available: R1: 7 R2: 4 R3: 5
----------------------------------------------------------
P2 | Available: R1: 7 R2: 5 R3: 5
----------------------------------------------------------
```

—---

Exercise no : 7(a) First Fit Allocation

```
#include <stdio.h>
#define MAX 100

int main() {
    int holes, processes;
    int holeSize[MAX], holeAvailable[MAX], processSize[MAX], allocation[MAX];
```

```
printf("Enter the number of Holes: ");
scanf("%d", &holes);

for (int i = 0; i < holes; i++) {
    printf("Enter size for hole H%d: ", i);
    scanf("%d", &holeSize[i]);
    holeAvailable[i] = holeSize[i];  // Initially, available = actual
}

printf("Enter number of processes: ");
scanf("%d", &processes);

for (int i = 0; i < processes; i++) {
    printf("Enter the size of process P%d: ", i);
    scanf("%d", &processSize[i]);
    allocation[i] = -1;  // -1 means not allocated
}

// First Fit Algorithm
for (int i = 0; i < processes; i++) {
    for (int j = 0; j < holes; j++) {
        if (holeAvailable[j] >= processSize[i]) {
            allocation[i] = j;  // allocate to hole j
            holeAvailable[j] -= processSize[i];
            break;
        }
    }
}

printf("\nFirst fit\n\n");
printf("Process\tSize\tHole\n");
for (int i = 0; i < processes; i++) {
    printf("P%d\t%d\t", i, processSize[i]);
    if (allocation[i] != -1)
        printf("H%d\n", allocation[i]);
    else
        printf("Not allocated\n");
```

```
    }

    printf("\nHole\tActual\tAvailable\n");
    for (int i = 0; i < holes; i++) {
        printf("H%d\t%d\t%d\n", i, holeSize[i], holeAvailable[i]);
    }

    return 0;
}
```

```
Enter number of holes: 5
Enter sizes of 5 holes: 100 500 200 300 600
Enter number of processes: 4
Enter sizes of 4 processes: 212 417 112 426

First Fit

Process PSize   Hole
P0      212     H1
P1      417     H4
P2      112     H1
P3      426     Not allocated

Hole    Actual  Available
H0      100     100
H1      500     176
H2      200     200
H3      300     300
H4      600     183
```

Exercise no : 7(b) Best Fit Allocation

```
#include <stdio.h>

#define MAX 100

int main() {
    int holes, processes;
    int holeSize[MAX], holeAvailable[MAX], processSize[MAX], allocation[MAX];

    printf("Enter the number of Holes: ");
    scanf("%d", &holes);
```

```
for (int i = 0; i < holes; i++) {
    printf("Enter size for hole H%d: ", i);
    scanf("%d", &holeSize[i]);
    holeAvailable[i] = holeSize[i];
}

printf("Enter number of processes: ");
scanf("%d", &processes);

for (int i = 0; i < processes; i++) {
    printf("Enter the size of process P%d: ", i);
    scanf("%d", &processSize[i]);
    allocation[i] = -1;
}

// Best Fit Algorithm
for (int i = 0; i < processes; i++) {
    int bestIdx = -1;
    for (int j = 0; j < holes; j++) {
        if (holeAvailable[j] >= processSize[i]) {
            if (bestIdx == -1 || holeAvailable[j] < holeAvailable[bestIdx]) {
                bestIdx = j;
            }
        }
    }

    if (bestIdx != -1) {
        allocation[i] = bestIdx;
        holeAvailable[bestIdx] -= processSize[i];
    }
}

printf("\nBest fit\n\n");
printf("Process\tSize\tHole\n");
for (int i = 0; i < processes; i++) {
    printf("P%d\t%d\t", i, processSize[i]);
    if (allocation[i] != -1)
```

```
            printf("H%d\n", allocation[i]);
        else
            printf("Not allocated\n");
    }

    printf("\nHole\tActual\tAvailable\n");
    for (int i = 0; i < holes; i++) {
        printf("H%d\t%d\t%d\n", i, holeSize[i], holeAvailable[i]);
    }

    return 0;
}
```

```
Enter the number of Holes: 5
Enter size for hole H0: 100
Enter size for hole H1: 500
Enter size for hole H2: 200
Enter size for hole H3: 300
Enter size for hole H4: 600
Enter number of processes: 4
Enter the size of process P0: 212
Enter the size of process P1: 417
Enter the size of process P2: 112
Enter the size of process P3: 426
```

```
Best fit

Process Size    Hole
P0   212 H3
P1   417 H1
P2   112 H2
P3   426 H4

Hole    Actual  Available
H0   100 100
H1   500 83
H2   200 88
H3   300 88
H4   600 174
```

Exercise no : 7(c) Worst Fit Allocation

```
#include <stdio.h>

#define MAX 100

int main() {
    int holes, processes;
    int holeSize[MAX], holeAvailable[MAX], processSize[MAX], allocation[MAX];
```

```
printf("Enter the number of Holes: ");
scanf("%d", &holes);

for (int i = 0; i < holes; i++) {
    printf("Enter size for hole H%d: ", i);
    scanf("%d", &holeSize[i]);
    holeAvailable[i] = holeSize[i];
}

printf("Enter number of processes: ");
scanf("%d", &processes);

for (int i = 0; i < processes; i++) {
    printf("Enter the size of process P%d: ", i);
    scanf("%d", &processSize[i]);
    allocation[i] = -1;
}

// Worst Fit Allocation
for (int i = 0; i < processes; i++) {
    int worstIdx = -1;
    for (int j = 0; j < holes; j++) {
        if (holeAvailable[j] >= processSize[i]) {
            if (worstIdx == -1 || holeAvailable[j] > holeAvailable[worstIdx]) {
                worstIdx = j;
            }
        }
    }

    if (worstIdx != -1) {
        allocation[i] = worstIdx;
        holeAvailable[worstIdx] -= processSize[i];
    }
}

printf("\nWorst fit\n\n");
printf("Process\tSize\tHole\n");
for (int i = 0; i < processes; i++) {
```

```
        printf("P%d\t%d\t", i, processSize[i]);
        if (allocation[i] != -1)
            printf("H%d\n", allocation[i]);
        else
            printf("Not allocated\n");
    }

    printf("\nHole\tActual\tAvailable\n");
    for (int i = 0; i < holes; i++) {
        printf("H%d\t%d\t%d\n", i, holeSize[i], holeAvailable[i]);
    }

    return 0;
}
```

```
Enter the number of Holes: 5
Enter size for hole H0: 100
Enter size for hole H1: 500
Enter size for hole H2: 200
Enter size for hole H3: 300
Enter size for hole H4: 600
Enter number of processes: 4
Enter the size of process P0: 212
Enter the size of process P1: 417
Enter the size of process P2: 112
Enter the size of process P3: 426
```

```
Worst fit

Process Size    Hole
P0  212 H4
P1  417 H1
P2  112 H4
P3  426 Not allocated

Hole    Actual  Available
H0  100 100
H1  500 83
H2  200 200
H3  300 300
H4  600 276
```

Exercise no : 8 Paging Technique

```
#include <stdio.h>
#include <math.h>

int main() {
    int process_size_kb, page_size = 4; // Assume page size = 4KB
    int num_pages, i, logical_address, page_number, offset;
    int page_table[10], frame_number, physical_address;

    printf("Enter process size (in KB, max 12KB): ");
    scanf("%d", &process_size_kb);

    num_pages = ceil((float)process_size_kb / page_size);
    printf("Total number of pages: %d\n", num_pages);

    printf("Enter page table values (frame number for each page):\n");
    for (i = 0; i < num_pages; i++) {
        printf("Page %d: ", i);
        scanf("%d", &page_table[i]);
    }

    printf("Enter relative (logical) address (in bytes): ");
    scanf("%d", &logical_address);

    page_number = logical_address / (page_size * 1024); // page size in bytes
    offset = logical_address % (page_size * 1024);

    if (page_number >= num_pages) {
        printf("Invalid logical address. Page number exceeds limit.\n");
        return 1;
    }

    frame_number = page_table[page_number];
    physical_address = frame_number * (page_size * 1024) + offset;

    printf("\nPage Table:\n");
```

```
    for (i = 0; i < num_pages; i++) {
        printf("%d [%d]\n", i, page_table[i]);
    }

    printf("\nPage Number = %d\n", page_number);
    printf("Offset = %d\n", offset);
    printf("Physical Address: %d\n", physical_address);

    return 0;
}
```

```
Enter process size (in KB, max 12KB): 12
Total number of pages: 3
Enter page table values (frame number for each page):
Page 0: 5
Page 1: 6
Page 2: 7
Enter relative (logical) address (in bytes): 2643

Page Table:
0 [5]
1 [6]
2 [7]

Page Number = 0
Offset = 2643
Physical Address: 23123
```

Exercise no : 9(a) FIFO page replacement

```
#include <stdio.h>
int main() {
    int length, frames, i, j, pageFaults = 0, position = 0;
    int referenceString[30], frameArray[10];
    printf("Enter length of ref. string: ");
    scanf("%d", &length);
    printf("Enter reference string (space-separated): ");
    for (i = 0; i < length; i++) {
        scanf("%d", &referenceString[i]);
    }
    printf("Enter number of frames: ");
```

```
    scanf("%d", &frames);
    if (frames <= 0) {
        printf("Number of frames must be greater than zero.\n");
        return 1;
    }
    for (i = 0; i < frames; i++) {
        frameArray[i] = -1;
    }
    printf("\nRef. str\tPage frames\n");
    for (i = 0; i < length; i++) {
        int page = referenceString[i];
        int hit = 0;
        for (j = 0; j < frames; j++) {
            if (frameArray[j] == page) {
                hit = 1;
                break;
            }
        }
        if (!hit) {
            frameArray[position] = page;
            position = (position + 1) % frames;
            pageFaults++;
        }
        printf("%d\t\t", page);
        for (j = 0; j < frames; j++) {
            if (frameArray[j] == -1)
                printf("- ");
            else
                printf("%d ", frameArray[j]);
        }
        printf("\n");
    }
    printf("\nTotal no. of page faults: %d\n", pageFaults);
    return 0;
}
```

```
Enter length of ref. string: 20
Enter reference string (space-separated): 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0
1
Enter number of frames: 3

Ref. str        Page frames
7               7 - -
0               7 0 -
1               7 0 1
2               2 0 1
0               2 0 1
3               2 3 1
0               2 3 0
4               4 3 0
2               4 2 0
3               4 2 3
0               0 2 3
3               0 2 3
2               0 2 3
1               0 1 3
2               0 1 2
0               0 1 2
1               0 1 2
7               7 1 2
0               7 0 2
1               7 0 1

Total no. of page faults: 15
```

Exercise no : 9(b) LRU page replacement

```
#include <stdio.h>

#define MAX_FRAMES 10
#define MAX_REFERENCES 50

int find_lru(int time[], int n) {
    int i, min = time[0], pos = 0;
    for (i = 1; i < n; i++) {
        if (time[i] < min) {
            min = time[i];
            pos = i;
        }
    }
    return pos;
}

int main() {
    int reference_string[MAX_REFERENCES], frames[MAX_FRAMES], time[MAX_FRAMES];
    int ref_length, num_frames, counter = 0, page_faults = 0;
    int i, j, pos;
```

```
    printf("Enter length of Reference string: ");
    scanf("%d", &ref_length);

    printf("Enter reference string: ");
    for (i = 0; i < ref_length; i++) {
        scanf("%d", &reference_string[i]);
    }

    printf("Enter number of frames: ");
    scanf("%d", &num_frames);

    for (i = 0; i < num_frames; i++) {
        frames[i] = -1;
        time[i] = 0;
    }

    printf("Ref. str Page frames\n");
    for (i = 0; i < ref_length; i++) {
        int page = reference_string[i];
        int found = 0;

        for (j = 0; j < num_frames; j++) {
            if (frames[j] == page) {
                found = 1;
                time[j] = counter++;
                break;
            }
        }

        if (!found) {
            if (i < num_frames) {
                pos = i;
            } else {
                pos = find_lru(time, num_frames);
            }
            frames[pos] = page;
            time[pos] = counter++;
            page_faults++;
```

```
        }

        printf("%d\t", page);
        for (j = 0; j < num_frames; j++) {
            if (frames[j] != -1)
                printf("%d ", frames[j]);
            else
                printf("-1");
        }
        printf("\n");
    }

    printf("Total Page Faults: %d\n", page_faults);
    return 0;
}
```

```
Enter length of Reference string: 20
Enter reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter number of frames: 3
Ref. str Page frames
7       7 --
0       7 0 -
1       7 0 1
2       2 0 1
0       2 0 1
3       2 0 3
0       2 0 3
4       4 0 3
2       4 0 2
3       4 3 2
0       0 3 2
3       0 3 2
2       0 3 2
1       1 3 2
2       1 3 2
0       1 0 2
1       1 0 2
7       1 0 7
0       1 0 7
1       1 0 7
Total Page Faults: 12
```

Exercise no : 9(c) Optimal page replacement

```
#include <stdio.h>
#define MAX 100
```

```
int findOptimal(int pages[], int frames[], int nf, int index, int l) {
    int farthest = -1, replaceIndex = -1;
    int i, j;

    for (i = 0; i < nf; i++) {
        for (j = index + 1; j < l; j++) {
            if (frames[i] == pages[j]) {
                if (j > farthest) {
                    farthest = j;
                    replaceIndex = i;
                }
                break;
            }
        }
        if (j == l) return i;
    }
    return (replaceIndex == -1) ? 0 : replaceIndex;
}

void printFrames(int frames[], int nf) {
    int i;
    for (i = 0; i < nf; i++) {
        if (frames[i] == -1)
            printf("   ");
        else
            printf("%2d ", frames[i]);
    }
    printf("\n");
}

int main() {
    int l, nf, count = 0;
    int i, j;

    printf("Length of Reference string: ");
    scanf("%d", &l);

    int pages[MAX];
    printf("Enter reference string:\n");
    for (i = 0; i < l; i++)
        scanf("%d", &pages[i]);

    printf("Enter number of frames: ");
```

```
    scanf("%d", &nf);

    int frames[MAX];
    for (i = 0; i < nf; i++)
        frames[i] = -1;

    printf("\nRef  Page frames\n");

    for (i = 0; i < l; i++) {
        int found = 0;

        for (j = 0; j < nf; j++) {
            if (frames[j] == pages[i]) {
                found = 1;
                break;
            }
        }

        printf("%2d ", pages[i]);
        if (found) {
            printf("\n");
            continue;
        }

        int emptyIndex = -1;
        for (j = 0; j < nf; j++) {
            if (frames[j] == -1) {
                emptyIndex = j;
                break;
            }
        }

        if (emptyIndex != -1) {
            frames[emptyIndex] = pages[i];
        } else {
            int replaceIndex = findOptimal(pages, frames, nf, i, l);
            frames[replaceIndex] = pages[i];
        }
        count++;
        printFrames(frames, nf);
    }
    printf("\nTotal no. of page faults: %d\n", count);
    return 0;
}
```

```
Length of Reference string: 20
Enter reference string:
1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6
Enter number of frames: 3

Ref. str  Page frames
 1  1
 2  1  2
 3  1  2  3
 4  1  2  4
 2
 1
 5  1  2  5
 6  1  2  6
 2
 1
 2
 3  3  2  6
 7  3  7  6
 6
 3
 2  3  2  6
 1  3  2  1
 2
 3
 6  6  2  1

Total no. of page faults: 11
```

Exercise no : 10 (a) Contiguous File Allocation

```
#include <stdio.h>
#include <stdbool.h>
#define DISK_SIZE 100
typedef struct {
    int start;
    int size;
    bool allocated;
} File;
void allocateFile(File files[], int *disk, int index, int start, int size) {
    bool isContiguous = true;
    int i;
    for (i = start; i < start + size; i++) {
        if (i >= DISK_SIZE || disk[i] == 1) {
            isContiguous = false;
            break;
```

```
        }
    }
    if (isContiguous) {
        for (i = start; i < start + size; i++) {
            disk[i] = 1;
        }
        files[index].start = start;
        files[index].size = size;
        files[index].allocated = true;
         printf("File %d allocated from block %d to %d\n", index + 1, start, start +
size - 1);
    } else {
        files[index].allocated = false;
        printf("File %d not allocated since %d contiguous memory not available from
%d\n", index + 1, start ,size);
    }
}
int main() {
    int disk[DISK_SIZE] = {0};
    int numFiles, i, start, size;
    printf("Enter the number of files: ");
    scanf("%d", &numFiles);
    File files[numFiles];
    for (i = 0; i < numFiles; i++) {
        printf("\nEnter starting block and size for file %d: ", i + 1);
        scanf("%d %d", &start, &size);

        allocateFile(files, disk, i, start, size);
    }
```

```
    printf("\nDisk allocation status:\n");
    for (i = 0; i < DISK_SIZE; i++) {
        printf("%d", disk[i]);
    }
    printf("\n");
    return 0;
}
```

```
Enter the number of files: 2

Enter starting block and size for file 1: 2 6
File 1 allocated from block 2 to 7

Enter starting block and size for file 2: 14 5
File 2 allocated from block 14 to 18

Disk allocation status:
00111111000000111110000000000000000000000000000000000000000000000000000000000000
```

—---

Exercise no : 10 (b) Indexed File Allocation

Option 1:

```
#include <stdio.h>
#include <string.h>

struct File {
    char name[20];
    int indexBlock;
    int length;
    int blocks[20];
};

int main() {
    int n, i, j;

    printf("Enter number of files: ");
    scanf("%d", &n);

    struct File files[n];
```

```
    for (i = 0; i < n; i++) {
        printf("Enter starting block and size of file%d: ", i + 1);
        scanf("%d %d", &files[i].indexBlock, &files[i].length);

        printf("Enter blocks occupied by file%d: ", i + 1);
        for (j = 0; j < files[i].length; j++) {
            scanf("%d", &files[i].blocks[j]);
        }
    }

    printf("\nFile\tIndex\tLength\n");
    for (i = 0; i < n; i++) {
        printf("%d \t %d \t %d \n", i + 1, files[i].indexBlock, files[i].length);
    }

    for (i = 0; i < n; i++) {
        printf("Enter file name for file%d: ", i + 1);
        scanf("%s", files[i].name);

        printf("file name is: %s\n", files[i].name);
        printf("Index is: %d\n", files[i].indexBlock);
        printf("Blocks occupied are: ");
        for (j = 0; j < files[i].length; j++) {
            printf("%d ", files[i].blocks[j]);
        }
        printf("\n\n");
    }
    return 0;
}
```

```
Enter number of files: 2
Enter starting block and size of file1: 5 3
Enter blocks occupied by file1: 5 6 7
Enter starting block and size of file2: 10 4
Enter blocks occupied by file2: 10 11 12 13

File    Index   Length
1        5       3
2        10      4
Enter file name for file1: file1
file name is: file1
Index is: 5
Blocks occupied are: 5 6 7

Enter file name for file2: file2
file name is: file2
Index is: 10
Blocks occupied are: 10 11 12 13
```

Option 2:

```
#include <stdio.h>

int main() {
   int numFiles, startBlock, size, i, j;

   printf("Enter no. of files: ");
   scanf("%d", &numFiles);

   int fileIndex[numFiles], fileLength[numFiles], blocks[numFiles][100];

   for (i = 0; i < numFiles; i++) {
      printf("\nEnter starting block and size of file%d: ", i + 1);
      scanf("%d %d", &startBlock, &size);

      fileIndex[i] = startBlock;
      fileLength[i] = size;

      printf("Enter blocks occupied by file%d: ", i + 1);
      for (j = 0; j < size; j++) {
         scanf("%d", &blocks[i][j]);
      }
   }

   printf("\nFile   Index   Length\n");
   for (i = 0; i < numFiles; i++) {
```

```
        printf("%d      %d       %d\n", i + 1, fileIndex[i], fileLength[i]);
    }

    printf("\nBlock occupied are:\n");
    for (i = 0; i < numFiles; i++) {
        for (j = 0; j < fileLength[i]; j++) {
            printf("%d ", blocks[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

```
Enter no. of files: 2

Enter starting block and size of file1: 2 5
Enter blocks occupied by file1: 2 5 4 6 7

Enter starting block and size of file2: 3 5
Enter blocks occupied by file2: 2 3 4 5 6

File    Index   Length
1       2       5
2       3       5

Block occupied are:
2 5 4 6 7
2 3 4 5 6
```

Option 3:

```
#include <stdio.h>
#include <stdlib.h>

#define DISK_SIZE 100  /* Total number of blocks in the disk */

typedef struct {
    int indexBlock;
    int size;
    int *blocks;
} File;

int disk[DISK_SIZE] = {0};  /* 0 means free, 1 means occupied */
```

```
/* Function to allocate blocks randomly */
void allocateFile(File *file, int fileIndex) {
    int i, count = 0;

    /* Allocate an index block */
    for (i = 0; i < DISK_SIZE; i++) {
        if (disk[i] == 0) {  /* Find a free block for index */
            file->indexBlock = i;
            disk[i] = 1;
            break;
        }
    }

    if (file->indexBlock == -1) {
        printf("File %d: No space available for index block\n", fileIndex + 1);
        return;
    }

    /* Allocate data blocks */
    file->blocks = (int *)malloc(file->size * sizeof(int));
    for (i = 0; i < DISK_SIZE && count < file->size; i++) {
        if (disk[i] == 0) {
            file->blocks[count] = i;
            disk[i] = 1;
            count++;
        }
    }

    if (count < file->size) {
        printf("File %d: Not enough free blocks available\n", fileIndex + 1);
        free(file->blocks);
        file->blocks = NULL;
    } else {
        printf("File %d allocated successfully!\n", fileIndex + 1);
    }
}

/* Function to display the File Allocation Table */
```

```
void displayFileTable(File files[], int numFiles) {
    int i, j;
    printf("\nFile Allocation Table:\n");
    printf("File No.\tIndex Block\tAllocated Blocks\n");
    for (i = 0; i < numFiles; i++) {
        if (files[i].blocks != NULL) {
            printf("%d\t\t%d\t\t", i + 1, files[i].indexBlock);
            for (j = 0; j < files[i].size; j++) {
                printf("%d ", files[i].blocks[j]);
            }
            printf("\n");
        } else {
            printf("%d\t\tNot Allocated\n", i + 1);
        }
    }
}

int main() {
    int numFiles, i;

    printf("Enter the number of files: ");
    scanf("%d", &numFiles);

    File files[numFiles];

    /* Input file sizes and allocate memory */
    for (i = 0; i < numFiles; i++) {
        printf("\nEnter the size (number of blocks) for File %d: ", i + 1);
        scanf("%d", &files[i].size);
        files[i].indexBlock = -1;
        files[i].blocks = NULL;
        allocateFile(&files[i], i);
    }

    /* Display file allocation table */
    displayFileTable(files, numFiles);

    /* Free allocated memory */
    for (i = 0; i < numFiles; i++) {
```

```
        if (files[i].blocks != NULL) {
            free(files[i].blocks);
        }
    }

    return 0;
}
```

```
Enter the number of files: 2

Enter the size (number of blocks) for File 1: 7
File 1 allocated successfully!

Enter the size (number of blocks) for File 2: 10
File 2 allocated successfully!

File Allocation Table:
File No.        Index Block     Allocated Blocks
1               0               1 2 3 4 5 6 7
2               8               9 10 11 12 13 14 15 16 17 18
```

Exercise no : 10 (c) Linked File Allocation

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_FILES 10
#define MAX_BLOCKS 100


typedef struct {
    char name[20];
    int startBlock;
    int size;
    int blocks[20];
} File;
```

```c
void allocateFile(File *file) {
    int i;

    printf("Enter file name: ");
    scanf("%s", file->name);

    printf("Enter starting block: ");
    scanf("%d", &file->startBlock);

    printf("Enter no. of blocks: ");
    scanf("%d", &file->size);

    printf("Enter block numbers: ");
    for (i = 0; i < file->size; i++) {
        scanf("%d", &file->blocks[i]);
    }
}

void displayFileTable(File files[], int numFiles) {
    int i, j;

    printf("\nFile\tStart\tSize\tBlock\n");
    for (i = 0; i < numFiles; i++) {
        printf("%s\t%d\t%d\t", files[i].name, files[i].startBlock, files[i].size);
        for (j = 0; j < files[i].size; j++) {
            printf("%d", files[i].blocks[j]);
            if (j < files[i].size - 1) {
                printf("-->");
            }
        }
        printf("\n");
    }
}

int main() {
    int numFiles, i;

    printf("Enter no. of files: ");
    scanf("%d", &numFiles);
```

```
    File files[MAX_FILES];

    for (i = 0; i < numFiles; i++) {
        allocateFile(&files[i]);
    }

    displayFileTable(files, numFiles);

    return 0;
}
```

```
Enter no. of files: 3
Enter file name: siva
Enter starting block: 10
Enter no. of blocks: 10
Enter block numbers: 5
4
8
7
6
1
2
3
9
0
Enter file name: bala
Enter starting block: 12
Enter no. of blocks: 5
Enter block numbers: 11
12
10
13
14
Enter file name: kumar
Enter starting block: 15
Enter no. of blocks: 5
Enter block numbers: 16
17
18
19
20

File    Start   Size    Block
siva    10      10      5-->4-->8-->7-->6-->1-->2-->3-->9-->0
bala    12      5       11-->12-->10-->13-->14
kumar   15      5       16-->17-->18-->19-->20
```

References:

https://www.geeksforgeeks.org/fcfs-disk-scheduling-algorithms/

Exercise no : 11(a) Disk Scheduling FCFS

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_REQUESTS 100

void FCFS(int request[], int n, int head) {
    int i;
    int seek_operations = 0;
    int distance, current_track;

    printf("\nSeek Sequence is:\n");

    for (i = 0; i < n; i++) {
        current_track = request[i];
        /* Calculate the absolute distance */
        distance = abs(current_track - head);
        seek_operations += distance;

        /* Move head to the current request */
        head = current_track;

        printf("%d\n", current_track);
    }

    printf("\nTotal number of seek operations = %d\n", seek_operations);
}

int main() {
    int request[MAX_REQUESTS];
    int n, head, i;

    /* Get the number of disk requests */
    printf("Enter number of requests (max %d): ", MAX_REQUESTS);
    scanf("%d", &n);

    if(n > MAX_REQUESTS || n <= 0) {
```

```
        printf("Invalid number of requests.\n");
        return 1;
    }

    /* Get the request sequence */
    printf("Enter the request sequence (space separated): ");
    for (i = 0; i < n; i++) {
        scanf("%d", &request[i]);
    }

    /* Get the initial head position */
    printf("Enter initial head position: ");
    scanf("%d", &head);

    /* Display the input */
    printf("\nInput:\n");
    printf("Request sequence = {");
    for (i = 0; i < n; i++) {
        printf("%d", request[i]);
        if (i < n - 1)
            printf(", ");
    }
    printf("}\nInitial head position = %d\n", head);

    /* Execute FCFS disk scheduling */
    printf("\nOutput:\n");
    FCFS(request, n, head);

    return 0;
}
```

```
Enter number of requests (max 100): 8
Enter the request sequence (space separated): 98 183 37 122 14 124 65 67
Enter initial head position: 53

Input:
Request sequence = {98, 183, 37, 122, 14, 124, 65, 67}
Initial head position = 53

Output:

Seek Sequence is:
98
183
37
122
14
124
65
67

Total number of seek operations = 640
```

Exercise no : 11(b) Disk Scheduling SSTF

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_REQUESTS 100

/* Function to find the nearest request */
int find_nearest(int request[], int processed[], int n, int head) {
    int i, min_distance = 99999, index = -1, distance;

    for (i = 0; i < n; i++) {
        if (!processed[i]) { /* If request is not yet processed */
            distance = abs(request[i] - head);
            if (distance < min_distance) {
                min_distance = distance;
                index = i;
            }
        }
    }
    return index;
}

void SSTF(int request[], int n, int head) {
```

```
    int i, seek_operations = 0, distance;
    int processed[MAX_REQUESTS] = {0}; /* Tracks processed requests */
    int completed = 0, index;

    printf("\nSeek Sequence is:\n");

    while (completed < n) {
        index = find_nearest(request, processed, n, head);
        if (index == -1) {
            break;
        }

        distance = abs(request[index] - head);
        seek_operations += distance;
        head = request[index];
        processed[index] = 1; /* Mark as processed */

        printf("%d\n", request[index]);
        completed++;
    }

    printf("\nTotal number of seek operations = %d\n", seek_operations);
}

int main() {
    int request[MAX_REQUESTS];
    int n, head, i;

    /* Get the number of disk requests */
    printf("Enter number of requests (max %d): ", MAX_REQUESTS);
    scanf("%d", &n);

    if (n > MAX_REQUESTS || n <= 0) {
        printf("Invalid number of requests.\n");
        return 1;
    }

    /* Get the request sequence */
    printf("Enter the request sequence (space-separated): ");
    for (i = 0; i < n; i++) {
        scanf("%d", &request[i]);
    }

    /* Get the initial head position */
```

```
    printf("Enter initial head position: ");
    scanf("%d", &head);

    /* Display input */
    printf("\nInput:\n");
    printf("Request sequence = {");
    for (i = 0; i < n; i++) {
        printf("%d", request[i]);
        if (i < n - 1)
            printf(", ");
    }
    printf("}\nInitial head position = %d\n", head);

    /* Execute SSTF Disk Scheduling */
    printf("\nOutput:\n");
    SSTF(request, n, head);

    return 0;
}
```

```
Enter number of requests (max 100): 8
Enter the request sequence (space-separated): 98 183 37 122 14 124 65 67
Enter initial head position: 53

Input:
Request sequence = {98, 183, 37, 122, 14, 124, 65, 67}
Initial head position = 53

Output:

Seek Sequence is:
65
67
37
14
98
122
124
183

Total number of seek operations = 236
```

Exercise no : 11(c) Disk Scheduling SCAN

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_REQUESTS 100
void SCAN(int request[], int n, int head) {
    int left[MAX_REQUESTS], right[MAX_REQUESTS];
    int lsize = 0, rsize = 0, seek_operations = 0;
    int i, j, distance, current_track, temp;
    for (i = 0; i < n; i++) {
        if (request[i] < head)
            left[lsize++] = request[i];
        else
            right[rsize++] = request[i];
    }
    for (i = 0; i < lsize - 1; i++) {
        for (j = i + 1; j < lsize; j++) {
            if (left[i] < left[j]) {
                temp = left[i];
                left[i] = left[j];
                left[j] = temp;
            }
        }
    }
    for (i = 0; i < rsize - 1; i++) {
        for (j = i + 1; j < rsize; j++) {
            if (right[i] > right[j]) {
                temp = right[i];
                right[i] = right[j];
                right[j] = temp;
            }
        }
    }
    printf("\nSeek Sequence:\n");
    for (i = 0; i < lsize; i++) {
        current_track = left[i];
        distance = abs(current_track - head);
        seek_operations += distance;
        head = current_track;
```

```
        printf("%d ", current_track);
    }
    distance = abs(0 - head);
    seek_operations += distance;
    head = 0;
    printf("%d ", head);
    for (i = 0; i < rsize; i++) {
        current_track = right[i];
        distance = abs(current_track - head);
        seek_operations += distance;
        head = current_track;
        printf("%d ", current_track);
    }
    printf("\n\nTotal Seek Operations: %d\n", seek_operations);
}
int main() {
    int request[MAX_REQUESTS], n, head, i;
    printf("Enter number of requests (max %d): ", MAX_REQUESTS);
    scanf("%d", &n);
    if(n > MAX_REQUESTS || n <= 0) {
        printf("Invalid number of requests.\n");
        return 1;
    }
    printf("Enter the request sequence (space-separated): ");
    for (i = 0; i < n; i++)
        scanf("%d", &request[i]);
    printf("Enter initial head position: ");
    scanf("%d", &head);
    printf("\nInput:\nRequest sequence = {");
    for (i = 0; i < n; i++) {
        printf("%d", request[i]);
        if (i < n - 1)
            printf(", ");
    }
    printf("}\nInitial head position = %d\n", head);
    printf("\nOutput:\n");
    SCAN(request, n, head);
    return 0;
}
```

```
Enter number of requests (max 100): 8
Enter the request sequence (space-separated): 98 183 37 122 14 124 65 67
Enter initial head position: 53

Input:
Request sequence = {98, 183, 37, 122, 14, 124, 65, 67}
Initial head position = 53

Output:

Seek Sequence:
37 14 0 65 67 98 122 124 183

Total Seek Operations: 236
```

Exercise no : 11(d) Disk Scheduling C-SCAN

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_REQUESTS 100
#define DISK_SIZE 199

void CSCAN(int request[], int n, int head) {
    int left[MAX_REQUESTS], right[MAX_REQUESTS];
    int lsize = 0, rsize = 0, seek_operations = 0;
    int i, j, temp, distance, current_track;
    for (i = 0; i < n; i++) {
        if (request[i] >= head)
            right[rsize++] = request[i];
        else
            left[lsize++] = request[i];
    }
    for (i = 0; i < rsize - 1; i++) {
        for (j = i + 1; j < rsize; j++) {
            if (right[i] > right[j]) {
                temp = right[i];
                right[i] = right[j];
                right[j] = temp;
            }
        }
    }
    for (i = 0; i < lsize - 1; i++) {
        for (j = i + 1; j < lsize; j++) {
```

```
            if (left[i] > left[j]) {
                temp = left[i];
                left[i] = left[j];
                left[j] = temp;
            }
        }
    }
    printf("\nSeek Sequence:\n");
    for (i = 0; i < rsize; i++) {
        current_track = right[i];
        distance = abs(current_track - head);
        seek_operations += distance;
        head = current_track;
        printf("%d ", current_track);
    }
    seek_operations += abs(DISK_SIZE - head);
    head = DISK_SIZE;
    printf("%d ", head);
    head = 0;
    seek_operations += DISK_SIZE;
    printf("%d ", head);
    for (i = 0; i < lsize; i++) {
        current_track = left[i];
        distance = abs(current_track - head);
        seek_operations += distance;
        head = current_track;
        printf("%d ", current_track);
    }
    printf("\n\nTotal Seek Operations: %d\n", seek_operations);
}

int main() {
    int request[MAX_REQUESTS], n, head, i;
    printf("Enter number of requests (max %d): ", MAX_REQUESTS);
    scanf("%d", &n);
    if (n > MAX_REQUESTS || n <= 0) {
        printf("Invalid number of requests.\n");
        return 1;
    }
    printf("Enter the request sequence (space-separated): ");
    for (i = 0; i < n; i++)
        scanf("%d", &request[i]);
    printf("Enter initial head position: ");
    scanf("%d", &head);
```

```c
    printf("\nInput:\nRequest sequence = {");
    for (i = 0; i < n; i++) {
        printf("%d", request[i]);
        if (i < n - 1)
            printf(", ");
    }
    printf("}\nInitial head position = %d\n", head);
    printf("\nOutput:\n");
    CSCAN(request, n, head);
    return 0;
}
```

```
Enter number of requests (max 100): 8
Enter the request sequence (space-separated): 98 183 37 122 14 124 65 67
Enter initial head position: 53

Input:
Request sequence = {98, 183, 37, 122, 14, 124, 65, 67}
Initial head position = 53

Output:

Seek Sequence:
65 67 98 122 124 183 199 0 14 37

Total Seek Operations: 382
```