1. Shell – Student DB

```
#!/bin/bash

db_name="student_database.txt"

create_data() {
   if [[ -e "$db_name" ]]; then
      echo "Database already exists!"
   else
      touch "$db_name"
      echo "Database created successfully!"
   fi
}

view_database() {
   if [[ -s "$db_name" ]]; then
      echo "--------------------------------------"
      printf "%-10s %-15s %-10s %-10s\n" "Roll_No" "Name" "Marks" "Result"
      echo "--------------------------------------"
      sort -n "$db_name" | while IFS=" " read -r roll name marks result; do
         printf "%-10s %-15s %-10s %-10s\n" "$roll" "$name" "$marks" "$result"
      done
      echo "--------------------------------------"
   else
      echo "Error: Database is empty!"
   fi
}

insert_data() {
   echo "Enter Roll Number:"
   read roll
```

```
    if grep -q "^$roll " "$db_name"; then
        echo "Error: Roll number '$roll' already exists!"
        return
    fi
    echo "Enter Name:"
    read name
    echo "Enter Marks:"
    read marks
    result=$(if [ "$marks" -ge 40 ]; then echo "Pass"; else echo "Fail"; fi)
    echo "$roll $name $marks $result" >> "$db_name"
    echo "Data inserted successfully!"
}

modify_data() {
    echo "Enter Roll Number to Modify:"
    read roll
    if grep -q "^$roll " "$db_name"; then
        echo "Enter New Name:"
        read new_name
        echo "Enter New Marks:"
        read new_marks
        new_result=$(if [ "$new_marks" -ge 40 ]; then echo "Pass"; else echo "Fail"; fi)
        sed -i "s/^$roll .*/$roll $new_name $new_marks $new_result/" "$db_name"
        echo "Record modified successfully!"
    else
        echo "Error: Roll number '$roll' not found!"
    fi
}

delete_data() {
    echo "Enter Roll Number to Delete:"
    read roll
    if grep -q "^$roll " "$db_name"; then
        sed -i "/^$roll /d" "$db_name"
        echo "Record deleted successfully!"
    else
        echo "Error: Roll number '$roll' not found!"
    fi
}

view_student() {
    echo "Enter Roll Number to View:"
    read roll
    if grep -q "^$roll " "$db_name"; then
        echo "--------------------------------------"
        printf "%-10s %-15s %-10s %-10s\n" "Roll_No" "Name" "Marks" "Result"
        echo "--------------------------------------"
        grep "^$roll " "$db_name" | while IFS=" " read -r roll name marks result; do
            printf "%-10s %-15s %-10s %-10s\n" "$roll" "$name" "$marks" "$result"
        done
        echo "--------------------------------------"
    else
        echo "Error: Roll number '$roll' not found!"
    fi
}

while true; do
    echo "1. Create Database"
    echo "2. View Database"
    echo "3. Insert Data"
    echo "4. Modify Data"
    echo "5. Delete Data"
    echo "6. View Result of Student"
    echo "7. Exit"
    echo "Enter your choice:"
    read choice
    case $choice in
        1) create_data ;;
        2) view_database ;;
        3) insert_data ;;
        4) modify_data ;;
        5) delete_data ;;
        6) view_student ;;
        7) exit 0 ;;
        *) echo "Invalid choice!" ;;
    esac
done
```

2. Fork

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

int main()
{
    int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int pid = fork();

    if (pid == 0)
    {
        int product = 1;
        for (int i = 0; i < 10; i++)
        {
            product *= arr[i];
        }
        printf("Child Process: Product = %d\n", product);
    }
    else
    {
        int sum = 0;
        for (int i = 0; i < 10; i++)
        {
            sum += arr[i];
        }
        printf("Parent Process: Sum = %d\n", sum);
    }

    return 0;
}
```

3. CPU

FCFS

```c
#include <stdio.h>

typedef struct {
    int pid, arrivalTime, burstTime;
    int completionTime, turnaroundTime, waitingTime;
} Process;

void FCFS(Process p[], int n) {
    int currentTime = 0;
    float totalWT = 0, totalTAT = 0;

    // Sort by arrival time
    for (int i = 0; i < n-1; i++)
        for (int j = i+1; j < n; j++)
            if (p[i].arrivalTime > p[j].arrivalTime) {
                Process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }

    for (int i = 0; i < n; i++) {
        if (currentTime < p[i].arrivalTime)
            currentTime = p[i].arrivalTime;
        p[i].completionTime = currentTime + p[i].burstTime;
        p[i].turnaroundTime = p[i].completionTime - p[i].arrivalTime;
        p[i].waitingTime = p[i].turnaroundTime - p[i].burstTime;
        currentTime = p[i].completionTime;
        totalWT += p[i].waitingTime;
        totalTAT += p[i].turnaroundTime;
    }

    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrivalTime,
p[i].burstTime, p[i].completionTime, p[i].turnaroundTime, p[i].waitingTime);

    printf("\nAvg WT = %.2f\nAvg TAT = %.2f\n", totalWT/n, totalTAT/n);
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    Process p[n];
    for (int i = 0; i < n; i++) {
        printf("P%d Arrival Time: ", i + 1);
        scanf("%d", &p[i].arrivalTime);
        printf("P%d Burst Time: ", i + 1);
        scanf("%d", &p[i].burstTime);
        p[i].pid = i + 1;
    }
    FCFS(p, n);
    return 0;
}
```

PRIORITY (NON PREEM) –

```c
#include <stdio.h>

typedef struct {
    int pid, arrivalTime, burstTime, priority;
    int completionTime, turnaroundTime, waitingTime;
} Process;

void PriorityNonPreemptive(Process p[], int n) {
    int completed = 0, currentTime = 0;
    float totalWT = 0, totalTAT = 0;
    int isCompleted[n];
    for (int i = 0; i < n; i++) isCompleted[i] = 0;

    while (completed != n) {
        int idx = -1, highest = 9999;
        for (int i = 0; i < n; i++)
            if (p[i].arrivalTime <= currentTime && !isCompleted[i] &&
p[i].priority < highest) {
                highest = p[i].priority;
                idx = i;
            }

        if (idx != -1) {
            currentTime += p[idx].burstTime;
            p[idx].completionTime = currentTime;
            p[idx].turnaroundTime = p[idx].completionTime -
p[idx].arrivalTime;
```

```
            p[idx].waitingTime = p[idx].turnaroundTime - p[idx].burstTime;
            isCompleted[idx] = 1;
            totalWT += p[idx].waitingTime;
            totalTAT += p[idx].turnaroundTime;
            completed++;
        } else {
            currentTime++;
        }
    }

    printf("\nPID\tAT\tBT\tPR\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrivalTime,
p[i].burstTime, p[i].priority, p[i].completionTime, p[i].turnaroundTime,
p[i].waitingTime);

    printf("\nAvg WT = %.2f\nAvg TAT = %.2f\n", totalWT/n, totalTAT/n);
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    Process p[n];
    for (int i = 0; i < n; i++) {
        printf("P%d Arrival Time: ", i + 1);
        scanf("%d", &p[i].arrivalTime);
        printf("P%d Burst Time: ", i + 1);
        scanf("%d", &p[i].burstTime);
        printf("P%d Priority (Lower = Higher): ", i + 1);
        scanf("%d", &p[i].priority);
        p[i].pid = i + 1;
    }
    PriorityNonPreemptive(p, n);
    return 0;
}
```

RR (PREEM) –

```
#include <stdio.h>

typedef struct {
    int pid, arrivalTime, burstTime, remainingTime;
    int completionTime, turnaroundTime, waitingTime;
} Process;

void RoundRobin(Process p[], int n, int quantum) {
```

```c
    int time = 0, completed = 0;
    float totalWT = 0, totalTAT = 0;
    for (int i = 0; i < n; i++)
        p[i].remainingTime = p[i].burstTime;

    while (completed != n) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (p[i].remainingTime > 0 && p[i].arrivalTime <= time) {
                done = 0;
                if (p[i].remainingTime > quantum) {
                    time += quantum;
                    p[i].remainingTime -= quantum;
                } else {
                    time += p[i].remainingTime;
                    p[i].completionTime = time;
                    p[i].turnaroundTime = p[i].completionTime -
p[i].arrivalTime;
                    p[i].waitingTime = p[i].turnaroundTime - p[i].burstTime;
                    totalWT += p[i].waitingTime;
                    totalTAT += p[i].turnaroundTime;
                    p[i].remainingTime = 0;
                    completed++;
                }
            }
        }
        if (done)
            time++;
    }

    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrivalTime,
p[i].burstTime, p[i].completionTime, p[i].turnaroundTime, p[i].waitingTime);

    printf("\nAvg WT = %.2f\nAvg TAT = %.2f\n", totalWT/n, totalTAT/n);
}

int main() {
    int n, quantum;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    Process p[n];
    for (int i = 0; i < n; i++) {
        printf("P%d Arrival Time: ", i + 1);
        scanf("%d", &p[i].arrivalTime);
        printf("P%d Burst Time: ", i + 1);
        scanf("%d", &p[i].burstTime);
        p[i].pid = i + 1;
    }
    printf("Enter Time Quantum: ");
    scanf("%d", &quantum);
    RoundRobin(p, n, quantum);
    return 0;
}
```

SJF (Preem) –

```c
#include <stdio.h>
#include <limits.h>

typedef struct {
    int pid, arrivalTime, burstTime, remainingTime;
    int completionTime, turnaroundTime, waitingTime;
} Process;

void SJF_Preemptive(Process p[], int n) {
    int completed = 0, currentTime = 0;
    float totalWT = 0, totalTAT = 0;

    for (int i = 0; i < n; i++)
        p[i].remainingTime = p[i].burstTime;

    while (completed != n) {
        int idx = -1, minRT = INT_MAX;
        for (int i = 0; i < n; i++)
            if (p[i].arrivalTime <= currentTime && p[i].remainingTime > 0 &&
p[i].remainingTime < minRT) {
                minRT = p[i].remainingTime;
                idx = i;
            }

        if (idx != -1) {
            p[idx].remainingTime--;
            currentTime++;
            if (p[idx].remainingTime == 0) {
                p[idx].completionTime = currentTime;
                p[idx].turnaroundTime = p[idx].completionTime -
p[idx].arrivalTime;
                p[idx].waitingTime = p[idx].turnaroundTime - p[idx].burstTime;
                totalWT += p[idx].waitingTime;
                totalTAT += p[idx].turnaroundTime;
                completed++;
            }
        } else {
            currentTime++;
        }
    }

    printf("\nPID\tAT\tBT\tCT\tTAT\tWT\n");
    for (int i = 0; i < n; i++)
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrivalTime,
p[i].burstTime, p[i].completionTime, p[i].turnaroundTime, p[i].waitingTime);

    printf("\nAvg WT = %.2f\nAvg TAT = %.2f\n", totalWT/n, totalTAT/n);
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    Process p[n];
    for (int i = 0; i < n; i++) {
        printf("P%d Arrival Time: ", i + 1);
        scanf("%d", &p[i].arrivalTime);
        printf("P%d Burst Time: ", i + 1);
        scanf("%d", &p[i].burstTime);
        p[i].pid = i + 1;
    }
    SJF_Preemptive(p, n);
    return 0;
}
```

4. MULTITHREADING –

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

#define MAX_THREADS 4

typedef struct
{
    int row;
    int col;
    int **matrix_a;
    int **matrix_b;
    int **result;
    int size;
} ThreadData;

void *multiply(void *arg)
```

```c
{
    ThreadData *data = (ThreadData *)arg;
    int sum = 0;

    for (int k = 0; k < data->size; k++)
    {
        sum += data->matrix_a[data->row][k] * data->matrix_b[k][data->col];
    }

    // Store the result in the thread data structure
    data->result[data->row][data->col] = sum;

    // Prepare the return value (sum)
    int *return_value = malloc(sizeof(int));
    *return_value = sum;

    pthread_exit(return_value);
}

int main()
{
    int size;
    printf("Enter the size of the square matrices: ");
    scanf("%d", &size);

    // Allocate memory for matrices
    int **matrix_a = (int **)malloc(size * sizeof(int *));
    int **matrix_b = (int **)malloc(size * sizeof(int *));
    int **result = (int **)malloc(size * sizeof(int *));

    for (int i = 0; i < size; i++)
    {
        matrix_a[i] = (int *)malloc(size * sizeof(int));
        matrix_b[i] = (int *)malloc(size * sizeof(int));
        result[i] = (int *)malloc(size * sizeof(int));
    }

    // Initialize matrices with sample values
    printf("Matrix A:\n");
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            matrix_a[i][j] = i + j;
            printf("%d ", matrix_a[i][j]);
        }
        printf("\n");
    }
    printf("\nMatrix B:\n");
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            matrix_b[i][j] = i - j;
            printf("%d ", matrix_b[i][j]);
        }
        printf("\n");
    }

    pthread_t threads[MAX_THREADS];
    ThreadData thread_data[MAX_THREADS];
    int thread_count = 0;
    int total_sum = 0;

    // Create threads to compute matrix multiplication
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            // Set up thread data
            thread_data[thread_count].row = i;
            thread_data[thread_count].col = j;
            thread_data[thread_count].matrix_a = matrix_a;
            thread_data[thread_count].matrix_b = matrix_b;
            thread_data[thread_count].result = result;
            thread_data[thread_count].size = size;

            // Create thread
            if (pthread_create(&threads[thread_count], NULL, multiply,
&thread_data[thread_count]))
            {
                fprintf(stderr, "Error creating thread\n");
                return 1;
            }

            thread_count++;

            // If we've reached max threads, wait for them to finish
            if (thread_count == MAX_THREADS)
            {
                for (int k = 0; k < MAX_THREADS; k++)
                {
                    int *thread_sum;
                    pthread_join(threads[k], (void **)&thread_sum);
                    total_sum += *thread_sum;
```

```c
                    free(thread_sum);
                }
                thread_count = 0;
            }
        }
    }

    // Wait for remaining threads to finish
    for (int k = 0; k < thread_count; k++)
    {
        int *thread_sum;
        pthread_join(threads[k], (void **)&thread_sum);
        total_sum += *thread_sum;
        free(thread_sum);
    }

    // Print the result matrix
    printf("\nResult Matrix:\n");
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }

    // Print the total sum of all elements
    printf("\nTotal sum of all elements: %d\n", total_sum);

    // Free allocated memory
    for (int i = 0; i < size; i++)
    {
        free(matrix_a[i]);
        free(matrix_b[i]);
        free(result[i]);
    }
    free(matrix_a);
    free(matrix_b);
    free(result);

    return 0;
}
```

5. Bankers –

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

```c
#define P 10
#define R 10

int available[R], max[P][R], allocation[P][R], need[P][R], work[R],
safeSeq[P], finish[P];
int numP, numR;

void inputDetails()
{
    printf("Enter number of processes: ");
    scanf("%d", &numP);
    printf("Enter number of resources: ");
    scanf("%d", &numR);

    printf("Enter Maximum Demand Matrix:\n");
    for (int i = 0; i < numP; i++)
        for (int j = 0; j < numR; j++)
            scanf("%d", &max[i][j]);

    printf("Enter Allocation Matrix:\n");
    for (int i = 0; i < numP; i++)
        for (int j = 0; j < numR; j++)
            scanf("%d", &allocation[i][j]);

    for (int i = 0; i < numP; i++)
        for (int j = 0; j < numR; j++)
            need[i][j] = max[i][j] - allocation[i][j];

    printf("Enter Available Resources:\n");
    for (int i = 0; i < numR; i++)
        scanf("%d", &available[i]);
}

void calculateSafeSequence()
{
    for (int i = 0; i < numP; i++)
        finish[i] = 0;

    for (int i = 0; i < numR; i++)
        work[i] = available[i];

    int count = 0;
    while (count < numP)
    {
        int found = 0;
        for (int i = 0; i < numP; i++)
        {
            if (!finish[i])
            {
                int j;
                for (j = 0; j < numR; j++)
                {
                    if (need[i][j] > work[j])
                        break;
                }
                if (j == numR)
                {
                    for (int k = 0; k < numR; k++)
                        work[k] += allocation[i][k];
                    safeSeq[count++] = i;
                    finish[i] = 1;
                    found = 1;
                }
            }
        }
        if (!found)
        {
            printf("System is in an unsafe state!\n");
            return;
        }
    }

    printf("Safe Sequence: ");
    for (int i = 0; i < numP; i++)
        printf("P%d ", safeSeq[i]);
    printf("\n");
}

void displayMatrices()
{
    printf("\nMaximum Matrix:\n");
    for (int i = 0; i < numP; i++)
    {
        for (int j = 0; j < numR; j++)
            printf("%d ", max[i][j]);
        printf("\n");
    }

    printf("\nAllocation Matrix:\n");
    for (int i = 0; i < numP; i++)
    {
        for (int j = 0; j < numR; j++)
            printf("%d ", allocation[i][j]);
        printf("\n");
    }
    printf("\nNeed Matrix:\n");
    for (int i = 0; i < numP; i++)
    {
        for (int j = 0; j < numR; j++)
            printf("%d ", need[i][j]);
        printf("\n");
    }

    printf("\nAvailable Resources: ");
    for (int i = 0; i < numR; i++)
        printf("%d ", available[i]);
    printf("\n");
}

int main()
{
    int choice;
    while (1)
    {
        printf("\n=== Banker's Algorithm Menu ===\n");
        printf("1. Input details\n");
        printf("2. Display matrices\n");
        printf("3. Find safe sequence\n");
        printf("4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            inputDetails();
            break;
        case 2:
            displayMatrices();
            break;
        case 3:
            calculateSafeSequence();
            break;
        case 4:
            return 0;
        default:
            printf("Invalid choice! Try again.\n");
        }
    }
}
```

6. PIPE –

1 –

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>

int main()
{
    int fd[2];
    char str[] = "Hello";
    char str2[10];
    pid_t x;
    pipe(fd);
    pipe(fd);
    x = fork();
    if (x == 0)
    {
        close(fd[0]);
        write(fd[1], str, strlen(str) + 1);
        close(fd[1]);
    }
    else
    {
        close(fd[1]);
        read(fd[0], str2, strlen(str) + 1);
        close(fd[0]);
        printf("msg=%s\n", str2);
    }
    return 0;
}
```

2 –

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <ctype.h>

void convert(char *str)
{
    while (*str != '\0')
    {
        *str = toupper(*str);
        str++;
    }
```

```c
}

int main()
{
    int fd1[2];
    int fd2[2];
    char buff[10];
    char buff2[10];
    char buff3[10];
    pid_t x;
    pipe(fd1);
    pipe(fd2);
    x = fork();
    if (x == 0)
    {
        close(fd1[1]);
        read(fd1[0], buff, 6);
        convert(buff2);
        close(fd1[0]);

        close(fd2[0]);
        write(fd2[1], buff2, 6);
        close(fd2[1]);
    }
    else
    {
        close(fd1[0]);
        read(fd1[1], "hello", 6);
        close(fd1[1]);

        close(fd2[1]);
        read(fd2[0], buff3, 6);
        printf("msg=%s\n", buff3);
    }
    return 0;
}
```

3 –

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
    pid_t x;
    int fd[2];
    int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int e;
    pipe(fd);
    x = fork();
    if (x == 0)
    {
        close(fd[1]);
        while ((read(fd[0], &e, sizeof(e))) > 0)
        {

            printf("e=%d\n", e);
        }
        close(fd[0]);
    }
    else
    {
        close(fd[0]);
        write(fd[1], a, sizeof(a));
        close(fd[1]);
    }
    return 0;
}
```

4 –

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
    pid_t x;
    int fd1[2], fd2[2];
    int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int e, sum = 0, result;
    pipe(fd1);
    pipe(fd2);
    x = fork();
    if (x == 0)
    {
        close(fd1[1]);
        while ((read(fd1[0], &e, sizeof(e))) > 0)
        {
            printf("e=%d\n", e);
            sum = sum + e;
        }

        close(fd1[0]);
        close(fd2[0]);
        write(fd2[1], &sum, sizeof(sum));
        close(fd2[1]);
    }
    else
    {
        close(fd1[0]);
        write(fd1[1], a, sizeof(a));
        close(fd1[1]);
        close(fd2[1]);
        while ((read(fd2[0], &result, sizeof(result))) > 0)
            printf("result=%d\n", result);
        close(fd2[0]);
    }
    return 0;
}
```

7. PAGE REPLACEMENT –

FIFO –

```
#include <stdio.h>

int main() {
    int frames, pages, page[50], temp[50], faults = 0;
    int i, j, k = 0, flag;

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

    printf("Enter the page reference string: ");
    for(i = 0; i < pages; i++)
        scanf("%d", &page[i]);

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

    for(i = 0; i < frames; i++)
        temp[i] = -1;

    printf("\nPage\tFrames\n");
    for(i = 0; i < pages; i++) {
        flag = 0;
        for(j = 0; j < frames; j++) {
            if(temp[j] == page[i]) {
                flag = 1;
                break;
            }
        }
        if(flag == 0) {
            temp[k] = page[i];
            k = (k + 1) % frames;
            faults++;

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

    printf("\nTotal Page Faults = %d\n", faults);
    return 0;
}
```

LRU –

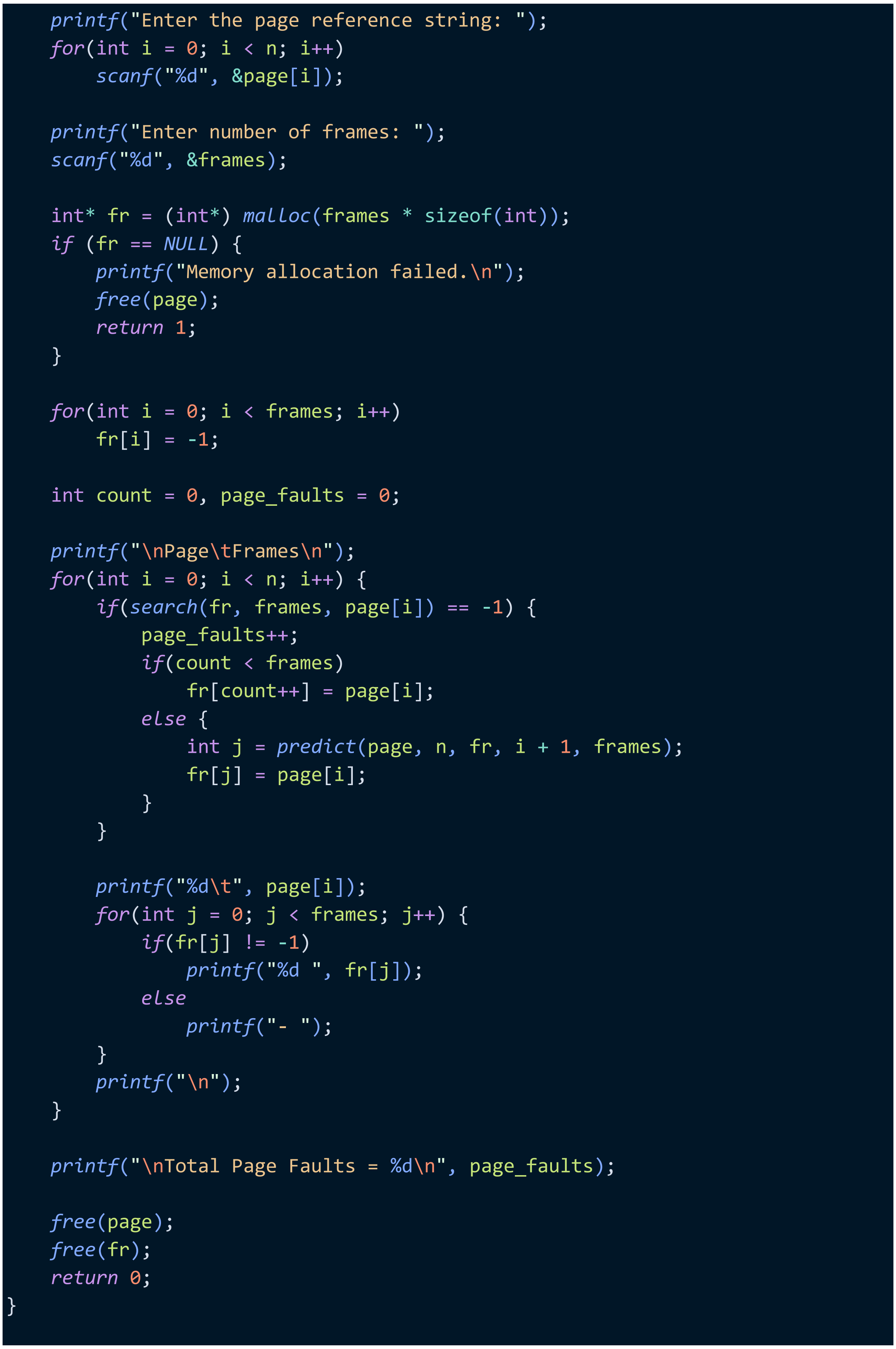
```
#include <stdio.h>

int main() {
    int frames, pages, page[50], temp[50], time[50], faults = 0;
    int i, j, pos, counter = 0, flag1, flag2, min;

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

    printf("Enter the page reference string: ");
    for(i = 0; i < pages; i++)
        scanf("%d", &page[i]);

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

    for(i = 0; i < frames; i++)
        temp[i] = -1;

    printf("\nPage\tFrames\n");
    for(i = 0; i < pages; i++) {
        flag1 = flag2 = 0;

        for(j = 0; j < frames; j++) {
            if(temp[j] == page[i]) {
```

```c
                counter++;
                time[j] = counter;
                flag1 = flag2 = 1;
                break;
            }
        }

        if(flag1 == 0) {
            for(j = 0; j < frames; j++) {
                if(temp[j] == -1) {
                    counter++;
                    faults++;
                    temp[j] = page[i];
                    time[j] = counter;
                    flag2 = 1;
                    break;
                }
            }
        }

        if(flag2 == 0) {
            pos = 0;
            min = time[0];
            for(j = 1; j < frames; j++) {
                if(time[j] < min) {
                    min = time[j];
                    pos = j;
                }
            }

            counter++;
            faults++;
            temp[pos] = page[i];
            time[pos] = counter;
        }

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

    printf("\nTotal Page Faults = %d\n", faults);
    return 0;
```

```
}
```

Optimal –

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

int search(int page[], int n, int key) {
    for(int i = 0; i < n; i++) {
        if(page[i] == key)
            return i;
    }
    return -1;
}

int predict(int page[], int n, int fr[], int index, int frames) {
    int res = -1, farthest = index;

    for(int i = 0; i < frames; i++) {
        int j;
        for(j = index; j < n; j++) {
            if(fr[i] == page[j]) {
                if(j > farthest) {
                    farthest = j;
                    res = i;
                }
                break;
            }
        }
        if(j == n)
            return i;
    }

    return (res == -1) ? 0 : res;
}

int main() {
    int n, frames;
    printf("Enter number of pages: ");
    scanf("%d", &n);

    int* page = (int*) malloc(n * sizeof(int));
    if (page == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }
```



```c
    printf("Enter the page reference string: ");
    for(int i = 0; i < n; i++)
        scanf("%d", &page[i]);

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

    int* fr = (int*) malloc(frames * sizeof(int));
    if (fr == NULL) {
        printf("Memory allocation failed.\n");
        free(page);
        return 1;
    }

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

    int count = 0, page_faults = 0;

    printf("\nPage\tFrames\n");
    for(int i = 0; i < n; i++) {
        if(search(fr, frames, page[i]) == -1) {
            page_faults++;
            if(count < frames)
                fr[count++] = page[i];
            else {
                int j = predict(page, n, fr, i + 1, frames);
                fr[j] = page[i];
            }
        }

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

    printf("\nTotal Page Faults = %d\n", page_faults);

    free(page);
    free(fr);
    return 0;
}
```