Shell Scripting

1. **Write a shell script to check number entered by the user is greater than 10.**

```
echo "Enter a number:"
read number

if [ "$number" -gt 10 ]; then
   echo "The number $number is greater than 10."
else
   echo "The number $number is not greater than 10."
fi
```

2. **Write a shell script to check if a file exists. If not, then create it.**

```
echo "Enter the file name (with path if necessary):"
read file_name

if [ -e "$file_name" ]; then
   echo "The file '$file_name' already exists."
else
   touch "$file_name"
   echo "File is not Exist. Created a new file"
   echo "The file '$file_name' has been created."
Fi
```

3. **Write a shell script that takes two command line arguments. Check whether the name passed as first argument is of a directory or not. If not, create directory using name passed as second argument.**

```
if [ $# -ne 2 ]; then
   echo "Usage: $0 <directory_to_check> <directory_to_create>"
   exit 1
fi

dir_to_check=$1
dir_to_create=$2

if [ -d "$dir_to_check" ]; then
   echo "'$dir_to_check' is already a directory."
else
```

```
    echo "'$dir_to_check' is not a directory."
    echo "Creating directory '$dir_to_create'..."
    mkdir "$dir_to_create"
    echo "Directory '$dir_to_create' has been created."
Fi
```

4. **Write a shell script which checks the total arguments passed. If the argument count is greater than 5, then display message "Too many arguments"**

```
#!/bin/bash

if [ $# -gt 5 ]; then
    echo "Too many arguments"
else
    echo "Argument count is acceptable: $# arguments provided."
Fi
```

5. **Write a shell script to check arguments passed at command line is whether of a file or directory.**

```
if [ $# -eq 0 ]; then
    echo "Usage: $0 <file_or_directory1> <file_or_directory2> ..."
    exit 1
fi

# Iterate through all the arguments
for item in "$@"; do
    if [ -d "$item" ]; then
        echo "'$item' is a directory."
    elif [ -f "$item" ]; then
        echo "'$item' is a file."
    else
        echo "'$item' is neither a file nor a directory."
    fi
done
```

6. **Write a shell script to read a month name from the user. Check if the name entered is either August or October.**

```
echo "Enter a month name:"
read month
```

```
# Convert the input to lowercase for case-insensitive comparison
month=$(echo "$month" | tr '[:upper:]' '[:lower:]')

if [ "$month" = "august" ]; then
   echo "The month entered is August."
elif [ "$month" = "october" ]; then
   echo "The month entered is October."
else
   echo "The month entered is neither August nor October."
fi
```

7. **Write a shell script to check whether file or directory exists.**

```
echo "Enter the file or directory name:"
read name

# Check if it exists
if [ -e "$name" ]; then
   if [ -f "$name" ]; then
      echo "'$name' exists and is a file."
   elif [ -d "$name" ]; then
      echo "'$name' exists and is a directory."
   else
      echo "'$name' exists but is neither a regular file nor a directory."
   fi
else
   echo "'$name' does not exist."
Fi
```

8. **Write a shell script to check whether file is exists and file is readable.**

```
echo "Enter the file name:"
read file_name

if [ -e "$file_name" ]; then
   # Check if it's a file
   if [ -f "$file_name" ]; then
      echo "'$file_name' exists and is a file."

      if [ -r "$file_name" ]; then
         echo "'$file_name' is readable."
      else
         echo "'$file_name' is not readable."
      fi
```

```
    elif [ -d "$file_name" ]; then
        echo "'$file_name' exists but it is a directory, not a file."
    else
        echo "'$file_name' exists but is neither a regular file nor a directory."
    fi
else
    echo "'$file_name' does not exist."
fi
```

9. Write a shell script to check if the present month is August   or not. Use date command to get present month.

```
current_month=$(date +%B)
# Check if the current month is August
if [ "$current_month" = "August" ]; then
    echo "The current month is August."
else
    echo "The current month is not August. It is $current_month."
fi
```

10. Write a shell script to check if the current user is root or regular user.

```
current_user=$(whoami)

# Check if the current user is root
if [ "$current_user" = "root" ]; then
    echo "The current user is root."
else
    echo "The current user is a regular user: $current_user."
Fi
```

11. Write a shell script to check the total arguments passed at command line. If the arguments are more than 3 then list the argument else print “type more next time”.

```
if [ $# -gt 3 ]; then
    echo "The arguments passed are:"
    # List all arguments
    for arg in "$@"; do
        echo "$arg"
    done
else
    echo "type more next time"
fi
```

1. **Write shell script to execute command ls, date, pwd repetitively.**

```
#!/bin/bash
repetitions=5
delay=2
for (( i=1; i<=repetitions; i++ ))
do
   echo "Iteration $i:"
   echo "Listing files:"
   ls
   echo "Current date and time:"
   date
   echo "Current working directory:"
   pwd
   echo "----------------------------"
   sleep $delay
done
echo "Script completed."
```

2. **Write a shell script to assign value to the variable? Display value with and without $.**

```
#!/bin/bash

my_variable="Hello, World!"

echo "Displaying value using \$:"

echo $my_variable

echo "Displaying value without \$:"

echo my_variable
```

OUTPUT:

3. **Variables are untyped in Shell Script. Write a shell script to show variables are untyped.**

```
#!/bin/bash
my_var="Hello, World!"
echo "Initially, my_var holds a string value: $my_var"
my_var=12345
echo "Now, my_var holds an integer value: $my_var"
my_var=3.14159
echo "Now, my_var holds a floating-point value: $my_var"
my_var=$(date)
echo "Now, my_var holds the output of a command: $my_var"
my_var=true
echo "Now, my_var holds a boolean-like value: $my_var"
```

OUTPUT:

4. Write a shell script to accept numbers from user. (Keyboard)

```
#!/bin/bash
echo "Enter a number:"
read num
if [[ "$num" =~ ^-?[0-9]+$ ]]; then
   echo "You entered an integer: $num"
elif [[ "$num" =~ ^-?[0-9]*\.[0-9]+$ ]]; then
   echo "You entered a floating-point number: $num"
else
   echo "The input is not a valid number."
fi
if [[ "$num" =~ ^-?[0-9]+$ || "$num" =~ ^-?[0-9]*\.[0-9]+$ ]]; then
   double=$(echo "$num * 2" | bc)
   echo "Double of the number is: $double"
fi
```

OUTPUT:

5. Write a shell script to accept numbers from command line arguments.

```
#!/bin/bash
if [ $# -eq 0 ]; then
   echo "Usage: $0 <number1> <number2> ... <numberN>"
   exit 1
fi
```

```
for num in "$@"; do
    if [[ "$num" =~ ^-?[0-9]+$ ]]; then
        echo "$num is an integer."
    elif [[ "$num" =~ ^-?[0-9]*\.[0-9]+$ ]]; then
        echo "$num is a floating-point number."
    else
        echo "$num is not a valid number."
        continue
    fi
    double=$(echo "$num * 2" | bc)
    echo "Double of $num is: $double"
done
```

6. **Write a shell script to show the contents of environmental variables SHELL, PATH, HOME.**

```
#!/bin/bash
echo "SHELL: $SHELL"
echo "PATH: $PATH"
echo "HOME: $HOME"
```

7. **Write a shell script to create two files. Accept file names from user.**

```
#!/bin/bash

# Prompt the user to enter the name for the first file
echo "Enter the name for the first file:"
read file1

# Prompt the user to enter the name for the second file
echo "Enter the name for the second file:"
read file2

# Create the files
touch "$file1"
touch "$file2"

# Check if files were created successfully
if [[ -f "$file1" && -f "$file2" ]]; then
    echo "Both files '$file1' and '$file2' have been created successfully."
else
    echo "There was an error creating the files."
fi
```

8. **Write a shell script to create two directories. Accept directories name from Command line.**

```
#!/bin/bash
if [ $# -ne 2 ]; then
   echo "Usage: $0 <directory1> <directory2>"
   exit 1
fi
dir1=$1
dir2=$2

mkdir "$dir1"
mkdir "$dir2"
if [ -d "$dir1" ] && [ -d "$dir2" ]; then
   echo "Both directories '$dir1' and '$dir2' have been created successfully."
else
   echo "There was an error creating the directories."
fi
```

9. **Write a shell script to copy file content of one file to another file. Accept files names from command line argument.**

```
#!/bin/bash
if [ $# -ne 2 ]; then
   echo "Usage: $0 <source_file> <destination_file>"
   exit 1
fi
source_file=$1
destination_file=$2
if [ ! -f "$source_file" ]; then
   echo "Source file '$source_file' does not exist."
   exit 1
fi
cp "$source_file" "$destination_file"
if [ $? -eq 0 ]; then
   echo "Content from '$source_file' has been copied to '$destination_file'."
else
   echo "There was an error copying the file content."
Fi
```

OUTPUT:

10. Write a shell script to rename the file name. Accept old filename and new filename from command line argument.

```
#!/bin/bash
if [ $# -ne 2 ]; then
   echo "Usage: $0 <old_filename> <new_filename>"
   exit 1
fi
old_filename=$1
new_filename=$2
if [ ! -f "$old_filename" ]; then
   echo "Error: '$old_filename' does not exist."
   exit 1
fi

mv "$old_filename" "$new_filename"
if [ $? -eq 0 ]; then
   echo "File '$old_filename' has been renamed to '$new_filename'."
else
   echo "Error: There was an issue renaming the file."
fi
```

11. Write a shell script to perform arithmetic operation of integer data.

```
#!/bin/bash
if [ $# -ne 3 ]; then
   echo "Usage: $0 <num1> <operator> <num2>"
   echo "Operators: +, -, *, /"
   exit 1
fi
num1=$1
operator=$2
num2=$3
if ! [[ "$num1" =~ ^-?[0-9]+$ ]] || ! [[ "$num2" =~ ^-?[0-9]+$ ]]; then
   echo "Error: Both arguments must be valid integers."
   exit 1
fi
case $operator in
   +)
      result=$((num1 + num2))
```

```
            ;;
        -)
            result=$((num1 - num2))
            ;;
        \*)
            result=$((num1 * num2))
            ;;
        /)
            # Check for division by zero
            if [ "$num2" -eq 0 ]; then
                echo "Error: Division by zero is not allowed."
                exit 1
            fi
            result=$((num1 / num2))
            ;;
        *)
            echo "Error: Invalid operator. Use one of +, -, *, /."
            exit 1
            ;;
    esac
    echo "Result: $num1 $operator $num2 = $result"
```

12.Write a shell script to perform arithmetic operation of float data.

```
if [ $# -ne 3 ]; then

    echo "Usage: $0 <num1> <operator> <num2>"

    echo "Operators: +, -, *, /"

    exit 1

fi

num1=$1

operator=$2

num2=$3

if ! [[ "$num1" =~ ^-?[0-9]+(\.[0-9]+)?$ ]] || ! [[ "$num2" =~ ^-?[0-9]+(\.[0-9]+)?$ ]]; then

    echo "Error: Both arguments must be valid floating-point numbers."

    exit 1
```

```
fi


if [ "$operator" == "/" ] && [ "$num2" == "0" ]; then

    echo "Error: Division by zero is not allowed."

    exit 1

fi

result=$(awk "BEGIN {print $num1 $operator $num2}")

echo "Result: $num1 $operator $num2 = $result"
```

Process Scheduling Algorithm

- **Write a C program to implement the First Come First Serve (Non-Pre-emptive) Algorithm.**

```
#include <stdio.h>

    int main()
    {
        int n, i;
        int arrivalTime[20], burstTime[20], waitingTime[20], turnaroundTime[20],
    completionTime[20];
        int totalWaitingTime = 0, totalTurnaroundTime = 0;

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

        for (i = 0; i < n; i++)
        {
            printf("Enter Arrival Time and Burst Time for Process %d: ", i + 1);
            scanf("%d%d", &arrivalTime[i], &burstTime[i]);
        }

        completionTime[0] = arrivalTime[0] + burstTime[0];
        for (i = 1; i < n; i++)
```

```
  {
        if (completionTime[i - 1] < arrivalTime[i])
       {
            completionTime[i] = arrivalTime[i] + burstTime[i];
        } else
        {
            completionTime[i] = completionTime[i - 1] + burstTime[i];
        }
    }



    for (i = 0; i < n; i++)
    {
        turnaroundTime[i] = completionTime[i] - arrivalTime[i];
    }

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

    printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround
Time\n");
    for (i = 0; i < n; i++)
    {
        totalWaitingTime += waitingTime[i];
        totalTurnaroundTime += turnaroundTime[i];
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, arrivalTime[i], burstTime[i],
waitingTime[i], turnaroundTime[i]);
    }

    printf("\nAverage Waiting Time: %.2f", (float)totalWaitingTime / n);
    printf("\nAverage Turnaround Time: %.2f\n", (float)totalTurnaroundTime / n);

    return 0;
}
```

- **Draw the Gantt charts and compute the finish time, turnaround time and waiting time for the following algorithms:**

  a. Priority scheduling.
  b. Shortest Job First (Non-Pre-emptive)

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

```
#include <stdlib.h>
#define MAX_PROCESSES 10
typedef struct {
    int id;
    int arrival_time;
    int priority;
    int burst_time;
    int finish_time;
    int turnaround_time;
    int waiting_time;
    int remaining_time;
} Process;

int compare_priority(const void* a, const void* b) {
    Process* process_a = (Process*)a;
    Process* process_b = (Process*)b;
    return process_a->priority - process_b->priority;
}

int compare_burst_time(const void* a, const void* b) {
    Process* process_a = (Process*)a;
    Process* process_b = (Process*)b;
    return process_a->burst_time - process_b->burst_time;
}

void priority_scheduling(Process processes[], int n) {
    qsort(processes, n, sizeof(Process), compare_priority);

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

        if (current_time < processes[i].arrival_time) {
            current_time = processes[i].arrival_time;
        }

        processes[i].finish_time = current_time + processes[i].burst_time;
        processes[i].turnaround_time = processes[i].finish_time - processes[i].arrival_time;
        processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
        current_time = processes[i].finish_time;
    }
}

void sjf_scheduling(Process processes[], int n) {
    qsort(processes, n, sizeof(Process), compare_burst_time);

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

        if (current_time < processes[i].arrival_time) {
            current_time = processes[i].arrival_time;
        }
        processes[i].finish_time = current_time + processes[i].burst_time;
        processes[i].turnaround_time = processes[i].finish_time - processes[i].arrival_time;
        processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
        current_time = processes[i].finish_time;
    }
}

void display_results(Process processes[], int n) {
    printf("Process ID\tArrival Time\tBurst Time\tFinish Time\tTurnaround
Time\tWaiting
Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id,
                processes[i].arrival_time, processes[i].burst_time,
                processes[i].finish_time, processes[i].turnaround_time,
                processes[i].waiting_time);
    }
}

int main() {
    int n;

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

    Process processes[MAX_PROCESSES];

    for (int i = 0; i < n; i++) {
        printf("Enter Process %d details:\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &processes[i].arrival_time);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burst_time);
        processes[i].id = i + 1;
        processes[i].priority = 0;
    }

    printf("\nPriority Scheduling Results:\n");
    priority_scheduling(processes, n);
    display_results(processes, n);

    printf("\nSJF Scheduling Results:\n");
    sjf_scheduling(processes, n);
    display_results(processes, n);

    return 0;
}
```

Draw the Gantt charts and compute the finish time, turnaround time and waiting time

for the following algorithms:

a. Round- Robin

```
#include <stdio.h>

struct process
{
int Pid;
int AT;
int BT;
int CT;
int TAT;
int WT;
int RT;
int remaining_BT;
};

void Sortarray(struct process ps[], int n)
{
struct process temp;
for (int i = 0; i < n; i++)
{
for (int j = i + 1; j < n; j++)
{
if (ps[i].AT > ps[j].AT)
{
temp = ps[i];
ps[i] = ps[j];
ps[j] = temp;
}
}
}
}
void roundRobin(struct process ps[], int n, int timeQuantum)
{
int time = 0;
int completedProcesses = 0;
int total_TAT = 0, total_WT = 0, total_RT = 0;
int total_idle_time = 0;

for (int i = 0; i < n; i++)
{
ps[i].remaining_BT = ps[i].BT;
ps[i].RT = -1;
}

while (completedProcesses < n)
{
int idle = 1;

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

```
if (ps[i].remaining_BT > 0 && ps[i].AT <= time)
{
idle = 0;

if (ps[i].RT == -1)
{
        ps[i].RT = time - ps[i].AT;
}

if (ps[i].remaining_BT > timeQuantum)
{
        time += timeQuantum;
ps[i].remaining_BT -= timeQuantum;

}
Else
{

 time += ps[i].remaining_BT;
ps[i].remaining_BT = 0;
ps[i].CT = time;
ps[i].TAT = ps[i].CT - ps[i].AT;
ps[i].WT = ps[i].TAT - ps[i].BT;

total_TAT += ps[i]. TAT;
total_WT += ps[i]. WT;
total_RT += ps[i]. RT;
completedProcesses++;

}
}
}

if (idle)
{
time++;
total_idle_time++;
}
}

printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\tRT\n");
for (int i = 0; i < n; i++)
{
printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", ps[i].Pid, ps[i].AT, ps[i].BT,
ps[i].CT, ps[i].TAT, ps[i].WT, ps[i].RT);
}

printf("\nAverage Turnaround Time: %.2f", (float)total_TAT / n);
printf("\nAverage Waiting Time: %.2f", (float)total_WT / n);
printf("\nAverage Response Time: %.2f", (float)total_RT / n);
```

```
int schedule_length = ps[n - 1].CT;
printf("\nThroughput = %.2f", (float)n / schedule_length);

float cpu_utilization = ((float)(schedule_length - total_idle_time) /
schedule_length) * 100;
printf("\nCPU Utilization = %.2f%%\n", cpu_utilization);
}

int main()
{


int n, timeQuantum;

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

if (n <= 0)
{
printf("Number of processes must be positive!\n");
return 1;
}
struct process ps[10];

for (int i = 0; i < n; i++)
{
ps[i].Pid = i + 1;
printf("Enter Arrival Time of Process %d: ", i + 1);
scanf("%d", &ps[i].AT);

if (ps[i].AT < 0)
{
printf("Arrival Time cannot be negative!\n");
return 1;
}

printf("Enter Burst Time of Process %d: ", i + 1);
scanf("%d", &ps[i].BT);

if (ps[i].BT <= 0)
{
printf("Burst Time must be positive!\n");
return 1;
}
}

printf("Enter the Time Quantum: ");
scanf("%d", &timeQuantum);

if (timeQuantum <= 0)
{
```

```
printf("Time Quantum must be positive!\n");
return 1;
}

Sortarray(ps, n);
roundRobin(ps, n, timeQuantum);

return 0;
}
```

- **Write a C program to implement the SRTF Scheduling Algorithm.**

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

void srtfPageReplacement(int pages[], int n, int frames) {
    int *frame = (int *)malloc(frames * sizeof(int));  // Array for frames
    int *nextUse = (int *)malloc(frames * sizeof(int));  // Next usage time for each frame
    int pageFaults = 0;

    // Initialize frames and nextUse array
    for (int i = 0; i < frames; i++) {
        frame[i] = -1;
    }

    for (int i = 0; i < n; i++) {
        int page = pages[i];
        int found = 0;

        // Check if the page is already in the frame
        for (int j = 0; j < frames; j++) {
            if (frame[j] == page) {
                found = 1;  // Page hit
                break;
            }
        }

        if (!found) {  // Page fault
            // Find the page with the farthest next use to replace
            int replaceIdx = -1;
            int farthest = -1;

            for (int j = 0; j < frames; j++) {
                int nextOccurrence = -1;

                // Find the next occurrence of the page in the future reference string
                for (int k = i + 1; k < n; k++) {
                    if (pages[k] == frame[j]) {
                        nextOccurrence = k;
                        break;
                    }
                }

                if (nextOccurrence == -1) {  // Page not used again in the future
                    replaceIdx = j;
                    break;
                }

                if (nextOccurrence > farthest) {  // Find the page with the farthest next use
                    farthest = nextOccurrence;
                    replaceIdx = j;
                }
            }

            // Replace the page
            frame[replaceIdx] = page;
            pageFaults++;
        }

        // Print the current page in the frame
        printf("Page Reference: %d -> Frames: ", page);
        for (int j = 0; j < frames; j++) {
            if (frame[j] == -1)
                printf(" - ");
            else
                printf("%d ", frame[j]);
        }
        printf("\n");
    }

    printf("\nTotal Page Faults: %d\n", pageFaults);

    // Free the dynamically allocated memory
    free(frame);
    free(nextUse);
}

int main() {
    int n, frames;

    // Get the number of pages from the user
    printf("Enter the number of pages: ");
    scanf("%d", &n);

    int *pages = (int *)malloc(n * sizeof(int));

    // Get the page reference string from the user
    printf("Enter the page reference string:\n");
    for (int i = 0; i < n; i++) {
        printf("Page %d: ", i + 1);
        scanf("%d", &pages[i]);
    }

    // Get the number of frames from the user
    printf("Enter the number of frames: ");
    scanf("%d", &frames);

    // Call the SRTF-like page replacement function
    srtfPageReplacement(pages, n, frames);

    // Free the dynamically allocated memory for pages
    free(pages);

    return 0;
}
```

- **Write a C program to implement the Priority(premptive) Scheduling Algorithm.**

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

typedef struct {
    int pid;       // Process ID
    int burstTime; // Burst Time
    int arrivalTime; // Arrival Time
    int priority;  // Priority
    int remainingTime; // Remaining Burst Time
    int completionTime; // Completion Time
    int turnAroundTime; // Turnaround Time
    int waitingTime;    // Waiting Time
} Process;

void calculateTimes(Process processes[], int n) {
    int currentTime = 0;
    int completed = 0;
    bool isProcessRunning = false;

    while (completed != n) {
        int highestPriorityIndex = -1;

        // Find the process with the highest priority that has arrived
        for (int i = 0; i < n; i++) {
            if (processes[i].arrivalTime <= currentTime && processes[i].remainingTime > 0)
{
                if (highestPriorityIndex == -1 || processes[i].priority <
processes[highestPriorityIndex].priority) {
                    highestPriorityIndex = i;
                }
            }
        }

        if (highestPriorityIndex != -1) {
            isProcessRunning = true;

            // Execute the process with the highest priority for one time unit
            processes[highestPriorityIndex].remainingTime--;
            currentTime++;

            // If the process is completed
            if (processes[highestPriorityIndex].remainingTime == 0) {
                completed++;
                processes[highestPriorityIndex].completionTime = currentTime;
                processes[highestPriorityIndex].turnAroundTime =
processes[highestPriorityIndex].completionTime -
processes[highestPriorityIndex].arrivalTime;
                processes[highestPriorityIndex].waitingTime =
processes[highestPriorityIndex].turnAroundTime -
processes[highestPriorityIndex].burstTime;
            }
        } else {
            // If no process is ready, move the time forward
            currentTime++;
        }
    }
}

void displayResults(Process processes[], int n) {
    float totalTurnAroundTime = 0, totalWaitingTime = 0;

    printf("\nProcess\tArrival\tBurst\tPriority\tCompletion\tTurnaround\tWaiting\n");
    for (int i = 0; i < n; i++) {
        totalTurnAroundTime += processes[i].turnAroundTime;
        totalWaitingTime += processes[i].waitingTime;

        printf("P%d\t%d\t%d\t%d\t\t%d\t\t%d\t\t%d\n",
               processes[i].pid,
               processes[i].arrivalTime,
               processes[i].burstTime,
               processes[i].priority,
               processes[i].completionTime,
               processes[i].turnAroundTime,
               processes[i].waitingTime);
    }
    printf("\nAverage Turnaround Time: %.2f", totalTurnAroundTime / n);
    printf("\nAverage Waiting Time: %.2f\n", totalWaitingTime / n);
}

int main() {
    int n;

    // Input the number of processes
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    Process processes[n];

    // Input process details
    for (int i = 0; i < n; i++) {
        processes[i].pid = i + 1;

        printf("Enter arrival time for Process %d: ", i + 1);
        scanf("%d", &processes[i].arrivalTime);

        printf("Enter burst time for Process %d: ", i + 1);
        scanf("%d", &processes[i].burstTime);

        printf("Enter priority for Process %d (lower value = higher priority): ", i + 1);
        scanf("%d", &processes[i].priority);

        // Initialize remaining time to burst time
        processes[i].remainingTime = processes[i].burstTime;
    }

    // Calculate times for each process
    calculateTimes(processes, n);

    // Display the results
    displayResults(processes, n);

    return 0;
}
```

- **Write a C program to implement the Bankers Algorithm for Deadlock Avoidance.**

```
#include <stdio.h>

#define MAX_PROCESSES 5
```

```
#define MAX_RESOURCES 3

int available[MAX_RESOURCES];
int maximum[MAX_PROCESSES][MAX_RESOURCES];
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
int safeSequence[MAX_PROCESSES];
int processCount, resourceCount;

int isSafe() {
    int work[MAX_RESOURCES];
    int finish[MAX_PROCESSES] = {0};
    int index = 0;

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

    printf("Process execution sequence:\n");

    int count = 0;
    while (count < processCount) {
        int found = 0;
        for (int p = 0; p < processCount; p++) {
            if (!finish[p]) {
                int canFinish = 1;
                for (int j = 0; j < resourceCount; j++) {
                    if (need[p][j] > work[j]) {
                        canFinish = 0;
                        break;
                    }
                }

                if (canFinish) {
                    for (int k = 0; k < resourceCount; k++)
                        work[k] += allocation[p][k];

                    safeSequence[index++] = p;
                    finish[p] = 1;
                    found = 1;
                    printf("P%d -> ", p);
                    count++;
                    break;
                }
            }
        }
        if (!found) {
            return 0;
        }
    }
    printf("End\n");
    return 1;
}

void calculateNeed() {
    for (int i = 0; i < processCount; i++)
        for (int j = 0; j < resourceCount; j++)
            need[i][j] = maximum[i][j] - allocation[i][j];
}

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

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

    printf("Enter the available resources for each type: ");
    for (int i = 0; i < resourceCount; i++)
        scanf("%d", &available[i]);

    printf("Enter the maximum resources for each process:\n");
    for (int i = 0; i < processCount; i++) {
        printf("Process %d: ", i);
        for (int j = 0; j < resourceCount; j++)
            scanf("%d", &maximum[i][j]);
    }

    printf("Enter the allocation for each process:\n");
    for (int i = 0; i < processCount; i++) {
        printf("Process %d: ", i);
        for (int j = 0; j < resourceCount; j++)
            scanf("%d", &allocation[i][j]);
    }

    calculateNeed();

    if (isSafe()) {
        printf("System is in a safe state.\nSafe sequence: ");
        for (int i = 0; i < processCount; i++)
            printf("P%d ", safeSequence[i]);
        printf("\n");
    } else {
        printf("System is not in a safe state.\n");
    }

    return 0;
}
```

Page Replacement Algorithms

Q.. Write a program to implement FIFO Page Replacement Algorithm.

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

void fifoPageReplacement(int frames[], int frameCount, int pages[], int pageCount) {
    int i, j, k, pageFaults = 0, pageHits = 0;
    int index = 0;

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

    printf("\nPage Reference String\tFrames\t\tPage Hit/Fault\n");

    for (i = 0; i < pageCount; i++) {
        int page = pages[i];
        int found = 0;

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

        if (found) {
            pageHits++;
            printf("%d\t\t\t", page);
            for (k = 0; k < frameCount; k++) {
                if (frames[k] != -1) {
                    printf("%d ", frames[k]);
                }
            }
            printf("\t\t\tHit\n");
        } else {
            pageFaults++;
            frames[index] = page;
            index = (index + 1) % frameCount;
            printf("%d\t\t\t", page);
            for (k = 0; k < frameCount; k++) {
                if (frames[k] != -1) {
                    printf("%d ", frames[k]);
                }
            }
            printf("\t\t\tFault\n");
        }
    }

    printf("\nTotal Page Hits: %d\n", pageHits);
    printf("Total Page Faults: %d\n", pageFaults);
}

int main() {
    int frameCount, pageCount, i;

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

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

    int* frames = (int*)malloc(frameCount * sizeof(int));
    int* pages = (int*)malloc(pageCount * sizeof(int));

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

    fifoPageReplacement(frames, frameCount, pages, pageCount);

    return 0;
}
```

- **Write a program to implement LRU Page Replacement Algorithm.**

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

int find_page(int frames[], int num_frames, int page) {
    for (int i = 0; i < num_frames; i++) {
        if (frames[i] == page) {
            return i;
        }
    }
    return -1;
}

int find_lru(int last_used[], int num_frames) {
    int lru_index = 0;
    for (int i = 1; i < num_frames; i++) {
        if (last_used[i] < last_used[lru_index]) {
            lru_index = i;
        }
    }
    return lru_index;
}

void lru_page_replacement(int pages[], int num_pages, int num_frames) {
    int* frames = (int*)malloc(num_frames * sizeof(int));
    int* last_used = (int*)malloc(num_frames * sizeof(int));
    int page_faults = 0;
    int page_hits = 0;
    int time = 0;

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

    printf("Page Reference\tFrames\t\tPage Fault/Hit\n");
    printf("--------------------------------------------------\n");

    for (int i = 0; i < num_pages; i++) {
        int page = pages[i];
        int index = find_page(frames, num_frames, page);

        if (index == -1) {
            page_faults++;
            int replace_index;


            if (page_faults <= num_frames) {
                replace_index = page_faults - 1;
            } else {
                replace_index = find_lru(last_used, num_frames);
            }

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

        } else {
            page_hits++;
            last_used[index] = time;

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

        time++;
    }

    printf("----------------------------------------------\n");
    printf("Total Frames: %d\n", num_frames);
    printf("Total Page Faults: %d\n", page_faults);
    printf("Total Page Hits: %d\n", page_hits);


    free(frames);
    free(last_used);
}

int main() {
    int num_frames, num_pages;

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

    printf("Enter the number of pages: ");
    scanf("%d", &num_pages);
    int* pages = (int*)malloc(num_pages * sizeof(int));

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

    lru_page_replacement(pages, num_pages, num_frames);

    free(pages);

    return 0;
}
```

- **Write a program to implement Optimal Page Replacement Algorithm.**

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

void optimalPageReplacement(int pages[], int n, int frames) {
    // Dynamically allocate memory for frame array
    int *frame = (int *)malloc(frames * sizeof(int));
    int count = 0, pageFaults = 0;

    // Initialize the frame array to -1 (empty)
    for (int i = 0; i < frames; i++) {
        frame[i] = -1;
    }

    printf("Page Reference String and Frame Status:\n");

    for (int i = 0; i < n; i++) {
        int page = pages[i];
        int found = 0;

        // Check if the page is already in the frame
        for (int j = 0; j < frames; j++) {
            if (frame[j] == page) {
                found = 1; // Page hit
                break;
            }
        }

        if (!found) { // Page fault
            if (count < frames) {
                frame[count] = page; // Fill the empty frames first
                count++;
            } else {
                // Find the page to replace
                int farthest = -1, replaceIdx = -1;
                for (int j = 0; j < frames; j++) {
                    int nextUse = -1;
                    for (int k = i + 1; k < n; k++) {
                        if (frame[j] == pages[k]) {
                            nextUse = k;
                            break;
                        }
                    }

                    if (nextUse == -1) { // Page not used in future
                        replaceIdx = j;
                        break;
                    } else if (nextUse > farthest) { // Farther use
                        farthest = nextUse;
                        replaceIdx = j;
                    }
                }
                frame[replaceIdx] = page;
            }
            pageFaults++;
        }

        // Print the current frame status
        printf("%2d -> ", page);
        for (int j = 0; j < frames; j++) {
            if (frame[j] == -1)
                printf(" - ");
            else
                printf("%2d ", frame[j]);
        }
        printf("\n");
    }

    printf("\nTotal Page Faults: %d\n", pageFaults);

    // Free the dynamically allocated memory
    free(frame);
}

int main() {
    int n, frames;
    // Get the number of pages and frames from the user
    printf("Enter the number of pages: ");
    scanf("%d", &n);
    int *pages = (int *)malloc(n * sizeof(int));

    printf("Enter the page reference string:\n");
    for (int i = 0; i < n; i++) {
        printf("Page %d: ", i + 1);
        scanf("%d", &pages[i]);
    }

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

    // Call the Optimal Page Replacement Algorithm
    optimalPageReplacement(pages, n, frames);

    // Free the dynamically allocated memory
    free(pages);

    return 0;
}
```

- **Write a program to implement Clock Page Replacement Algorithm.**

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

// Function for Least Recently Used (LRU) Page Replacement
int leastRecentlyUsed(int pages[], int n, int frames) {
    int *frame = (int *)malloc(frames * sizeof(int));
    int *lastUsed = (int *)malloc(frames * sizeof(int));
    int pageFaults = 0;

    for (int i = 0; i < frames; i++) {
        frame[i] = -1;
        lastUsed[i] = -1;
    }

    for (int i = 0; i < n; i++) {
        int page = pages[i];
        int found = 0;

        // Check if the page is already in the frame
        for (int j = 0; j < frames; j++) {
            if (frame[j] == page) {
                found = 1;
                lastUsed[j] = i; // Update last used time
                break;
            }
        }

        if (!found) { // Page fault
            int replaceIdx = -1;
            int leastRecentlyUsedTime = INT_MAX;

            // Find the least recently used frame
            for (int j = 0; j < frames; j++) {
                if (frame[j] == -1) { // Empty frame
                    replaceIdx = j;
                    break;
                } else if (lastUsed[j] < leastRecentlyUsedTime) {
                    leastRecentlyUsedTime = lastUsed[j];
                    replaceIdx = j;
                }
            }

            frame[replaceIdx] = page;
            lastUsed[replaceIdx] = i;
            pageFaults++;
        }
    }

    free(frame);
    free(lastUsed);

    return pageFaults;
}

// Function for Second Chance (Clock) Page Replacement
int secondChance(int pages[], int n, int frames) {
    int *frame = (int *)malloc(frames * sizeof(int));
    int *referenceBit = (int *)malloc(frames * sizeof(int));
    int pageFaults = 0, pointer = 0;

    for (int i = 0; i < frames; i++) {
        frame[i] = -1;
        referenceBit[i] = 0;
    }

    for (int i = 0; i < n; i++) {
        int page = pages[i];
        int found = 0;

        // Check if the page is already in the frame
        for (int j = 0; j < frames; j++) {
            if (frame[j] == page) {
                found = 1;
                referenceBit[j] = 1; // Set the reference bit
                break;
            }
        }

        if (!found) { // Page fault
            while (referenceBit[pointer] == 1) {
                referenceBit[pointer] = 0; // Reset reference bit
                pointer = (pointer + 1) % frames; // Move to the next frame
            }

            frame[pointer] = page;
            referenceBit[pointer] = 1;
            pointer = (pointer + 1) % frames; // Move pointer to the next frame
            pageFaults++;
        }
    }

    free(frame);
    free(referenceBit);

    return pageFaults;
}

int main() {
    int n, frames;

    // Get the number of pages and the page reference string from the user
    printf("Enter the number of pages: ");
    scanf("%d", &n);

    int *pages = (int *)malloc(n * sizeof(int));
    printf("Enter the page reference string:\n");
    for (int i = 0; i < n; i++) {
        printf("Page %d: ", i + 1);
        scanf("%d", &pages[i]);
    }

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

    // Least Recently Used (LRU)
    int lruFaults = leastRecentlyUsed(pages, n, frames);
    printf("\nTotal Page Faults using LRU: %d\n", lruFaults);

    // Second Chance (Clock)
    int secondChanceFaults = secondChance(pages, n, frames);
    printf("Total Page Faults using Second Chance: %d\n", secondChanceFaults);

    // Free the dynamically allocated memory
    free(pages);

    return 0;
}
```

Disk Scheduling Algorithms

1.SSTF

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

void SSTF(int requests[], int n, int head) {
    int total_seek_time = 0, completed = 0, min_distance, current, index;
     int *visited = (int *)malloc(n * sizeof(int));

    // Initialize the visited array
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }

    printf("\nSSTF Disk Scheduling\n");
    printf("Seek Sequence: %d", head);

    while (completed < n) {
        min_distance = INT_MAX; // Initialize to a large value
        index = -1;

        // Find the request with the minimum seek time
        for (int i = 0; i < n; i++) {
            if (!visited[i]) {
                int distance = abs(requests[i] - head);
                if (distance < min_distance) {
                    min_distance = distance;
                    index = i;
                }
            }
        }

        // Process the request with the minimum seek time
        visited[index] = 1;
        total_seek_time += min_distance;
        head = requests[index];
        printf(" -> %d", head);
        completed++;
    }

    printf("\nTotal Seek Time: %d\n", total_seek_time);
    printf("Average Seek Time: %.2f\n", (float)total_seek_time / n);
}

int main() {
    int n, head;

    printf("Enter the number of disk requests: ");
    scanf("%d", &n);

    // Dynamically allocate memory for the requests array
    int *requests = (int *)malloc(n * sizeof(int));
    if (requests == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    printf("Enter the disk requests (space-separated): ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

    printf("Enter the initial position of the disk head: ");
    scanf("%d", &head);

    SSTF(requests, n, head);

    // Free the allocated memory
    free(requests);

    return 0;
}
```

2. SCAN

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

void SCAN(int requests[], int n, int head, int disk_size, int direction) {
    int total_movement = 0;
    int i, j;
    int *sorted_requests = (int *)malloc((n + 1) * sizeof(int)); // Dynamic memory allocation
    int sorted_index = 0;

    // Copy the requests and include the head position for sorting
```

```
for (i = 0; i < n; i++) {
    sorted_requests[i] = requests[i];
}
sorted_requests[n] = head; // Add head position to the list
n++; // Increment size due to added head

// Sort the requests
for (i = 0; i < n - 1; i++) {
    for (j = i + 1; j < n; j++) {
        if (sorted_requests[i] > sorted_requests[j]) {
            int temp = sorted_requests[i];
            sorted_requests[i] = sorted_requests[j];
            sorted_requests[j] = temp;
        }
    }
}

// Find the position of the head in the sorted list
for (i = 0; i < n; i++) {
    if (sorted_requests[i] == head) {
        sorted_index = i;
        break;
    }
}

printf("\nSCAN Disk Scheduling (Direction: %s):\n", direction == 1 ? "Up" : "Down");
printf("Order of access: ");

// Move in the specified direction
if (direction == 1) { // Upward direction
    for (i = sorted_index; i < n; i++) {
        printf("%d ", sorted_requests[i]);
        if (i > sorted_index) {
            total_movement += abs(sorted_requests[i] - sorted_requests[i - 1]);
        }
    }
    if (sorted_requests[n - 1] != disk_size - 1) { // Move to the end of the disk
        total_movement += abs(disk_size - 1 - sorted_requests[n - 1]);
        printf("%d ", disk_size - 1);
    }
    for (i = sorted_index - 1; i >= 0; i--) { // Move downward
        printf("%d ", sorted_requests[i]);
        total_movement += abs(sorted_requests[i] - sorted_requests[i + 1]);
    }
} else { // Downward direction
    for (i = sorted_index; i >= 0; i--) {
            printf("%d ", sorted_requests[i]);
            if (i < sorted_index) {
                total_movement += abs(sorted_requests[i] - sorted_requests[i + 1]);
            }
        }
        if (sorted_requests[0] != 0) { // Move to the start of the disk
            total_movement += abs(sorted_requests[0]);
            printf("0 ");
        }
        for (i = sorted_index + 1; i < n; i++) { // Move upward
            printf("%d ", sorted_requests[i]);
            total_movement += abs(sorted_requests[i] - sorted_requests[i - 1]);
        }
    }

    printf("\nTotal head movement: %d\n", total_movement);

    // Free dynamically allocated memory
    free(sorted_requests);
}

int main() {
    int n, head, disk_size, direction;

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

    int *requests = (int *)malloc(n * sizeof(int)); // Dynamic memory allocation
    printf("Enter the requests: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

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

    printf("Enter disk size: ");
    scanf("%d", &disk_size);

    printf("Enter direction for SCAN (1 for up, 0 for down): ");
    scanf("%d", &direction);

    SCAN(requests, n, head, disk_size, direction);

    // Free dynamically allocated memory
    free(requests);
    return 0;
}
```

- **C-SCAN**

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

void CSCAN(int requests[], int n, int head, int disk_size) {
    int total_movement = 0;
    int i, j;
    int *sorted_requests = (int *)malloc((n + 3) * sizeof(int)); // Dynamic memory
allocation
    int sorted_index = 0;

    // Copy the requests and include the head position and disk boundaries
    for (i = 0; i < n; i++) {
        sorted_requests[i] = requests[i];
    }
    sorted_requests[n] = head;       // Add head position
    sorted_requests[n + 1] = 0;      // Add boundary at 0
    sorted_requests[n + 2] = disk_size - 1; // Add boundary at max disk size
    n += 3; // Increment size for the added values

    // Sort the requests
    for (i = 0; i < n - 1; i++) {
        for (j = i + 1; j < n; j++) {
            if (sorted_requests[i] > sorted_requests[j]) {
                int temp = sorted_requests[i];
                sorted_requests[i] = sorted_requests[j];
                sorted_requests[j] = temp;
            }
        }
    }

    // Find the position of the head in the sorted list
    for (i = 0; i < n; i++) {
        if (sorted_requests[i] == head) {
            sorted_index = i;
            break;
        }
    }

    printf("\nC-SCAN Disk Scheduling:\n");
    printf("Order of access: ");

    // Move upward and wrap around
    for (i = sorted_index; i < n; i++) {
        printf("%d ", sorted_requests[i]);
        if (i > sorted_index) {
            total_movement += abs(sorted_requests[i] - sorted_requests[i - 1]);
        }
    }
    if (sorted_requests[n - 1] != disk_size - 1) {
        total_movement += abs(disk_size - 1 - sorted_requests[n - 1]);
        printf("%d ", disk_size - 1);
    }
    total_movement += disk_size - 1; // Wrap around to 0
    printf("0 ");
    for (i = 1; i < sorted_index; i++) { // Continue upward from 0
        printf("%d ", sorted_requests[i]);
        total_movement += abs(sorted_requests[i] - sorted_requests[i - 1]);
    }

    printf("\nTotal head movement: %d\n", total_movement);

    // Free dynamically allocated memory
    free(sorted_requests);
}

int main() {
    int n, head, disk_size;

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

    int *requests = (int *)malloc(n * sizeof(int)); // Dynamic memory allocation
    printf("Enter the requests: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &requests[i]);
    }

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

    printf("Enter disk size: ");
```

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

        CSCAN(requests, n, head, disk_size);

        // Free dynamically allocated memory
        free(requests);

        return 0;
    }
```

- **OS Phase 1:**

```
#include <stdio.h>
#include <stdlib.h>
FILE *ptr , *wptr ;
int tempi = 0 , tempj = 0 , flag = 0 , C = 0 , IC = 00 , SI = 00 , ttl = 0 ;
char M[100][4] , IR[4] , R[4] , buffer[40] , ttl_array[4] ;
void init()
{
   int i = 0 , j = 0 ;
   for(i=8,j=0;i<12 && j<4;i++,j++)
      ttl_array[j] = buffer[i];
   ttl = atoi(ttl_array);//atoi : covrrt char numeric string to a integer
}
void reset()
{
   int i = 0 , j = 0 ;
   C = 00 , IC = 00 , ttl = 0 , tempi = 0 , tempj = 0 , flag = 0;
   for(i=0;i<4;i++)
   {
      IR[i] = '\0';
      R[i] = '\0';
      ttl_array[i] = '\0';
   }
   for(i=0;i<100;i++)
   {
      for(j=0;j<4;j++)
         M[i][j] = '\0';
   }
   printf("CPU Reseted Successfully!\n");
   init();
}
void loadbuffer()
{
   int i ;
   char ch ;
```

```
    for(i=0;i<40;i++)
        buffer[i] = '\0';
    i = 0 ;
    while((ch = getc(ptr)) != '\n')
    {
        if(i == 40)
            break;
        buffer[i] = ch ;
        i++;
    }
}
int check()
{
    if(buffer[0] == '$' && buffer[1] == 'A')
        return 1;//amj
    else if(buffer[0] == '$' && buffer[1] == 'D')
        return 2;//data
    else if(buffer[0] == '$' && buffer[1] == 'E')
        return 3;//end
    else
        return 4;//code
}
void terminate()
{
    printf("\nJob Terminated !");
}
void load()
{
    if(tempj==4)
        tempj = 0;
    int i = tempi , j = tempj , k = 0 ;
    if(i < 10 && j < 4 && flag == 0)
    {
        //block 0 reserved for code
        for(i=tempi;i<10;i++)
        {
            for(j=tempj;j<4;j++)
            {
                M[i][j] = buffer[k];
                k++;
            }
        }
    }
    if(i >= 10 && i < 100 && j < 4)
    {
        //block 1 to block 9
        for(i=tempi;i<100;i++)
        {
            for(j=tempj;j<4;j++)
            {
                if(k==40)
                    break;
                M[i][j] = buffer[k];
                k++;
            }
        }
    }
    tempi = i ;
    tempj = j ;
}
void GD(int operand)
{
    loadbuffer();
    int i = operand , j =  0 , k = 0 ;
    for(i = operand ; i < (operand+10) ; i++)
    {
        for(j=0;j<4;j++)
        {
            if(k==40)
                break;
            M[i][j] = buffer[k];
            k++;
        }
    }
}
void PD(int operand)
{
    int i = operand , j = 0  ;
    for(i = operand ; i < (operand+10) ; i++)
    {
        for(j=0;j<4;j++)
        {
            if(M[i][j] != '\0')
                fprintf(wptr,"%c",M[i][j]);
            else
                fprintf(wptr," ");
        }
    }
    fprintf(wptr,"\n");
}
void H()
{
    fprintf(wptr,"\n\n");
}
void LR(int operand)
{
    int i , j = 0 ;
    i = operand ;
    for(j = 0 ; j < 4 ; j++)
        R[j] = M[i][j];
}
void SR(int operand)
{
    int i , j ;
    i = operand;
    for(j=0;j<4;j++)
        M[i][j] = R[j];
}
void CR(int operand)
{
    int i = operand , j = 0 , counter = 0;
    for(j = 0 ; j < 4 ; j++)
    {
        if(R[j] == M[i][j])
            counter = counter+1;
    }
    if(counter == 4)
        C = 01;
}
void BT(int operand)
{
    if(C==01)
        IC = operand - 1 ;
    else
        IC = IC ;
}
void MOS(int op)
{
    switch (SI)
    {
        case 1:GD(op);
        break;
        case 2:PD(op);
        break;
        case 3:H();
        break;
    }
}
```

```
void execute()
{
   int i , j , k = 0 , op = 0; ;
   char operand[3] ;
   while(IC < ttl)
   {
      for(i=IC;i<IC+1;i++)
      {
         for(j=0;j<4;j++)
         {
            IR[k] = M[i][j];
            k++;
         }
      }
      for(i=2,j=0;i<4;i++,j++)
         operand[j] = IR[i];
      operand[2] = '\0';
      op = atoi(operand);
      if(IR[0] == 'G' && IR[1] == 'D')
      {   SI = 1 ; MOS(op); }
      else if(IR[0] == 'P' && IR[1] == 'D')
      {   SI = 2 ; MOS(op); }
      else if(IR[0] == 'L' && IR[1] == 'R')
         LR(op);
      else if(IR[0] == 'S' && IR[1] == 'R')
         SR(op);
      else if(IR[0] == 'C' && IR[1] == 'R')
         CR(op);
      else if(IR[0] == 'B' && IR[1] == 'T')
         BT(op);
      else if(IR[0] == 'H')
      {   SI = 3 ; MOS(op); }
      IC = IC + 01;
      k = 0 ;
   }
}
int main()
{
   int temp = 0 ;
   ptr = fopen("input.txt","r");
   wptr = fopen("output.txt","w");
   while(!feof(ptr))
   {
      loadbuffer();
      temp = check();
      if(temp == 1)
```

```
            reset();
        else if(temp == 2)
        {
            flag = 1;
            execute();
        }
        else if(temp == 3)
            terminate();
        else if(temp == 4)
            load();
    }
    printf("\nAll jobs executed!");
    fclose(ptr);
    fclose(wptr);
    return 0;
}
```

Input.txt

```
input.txt
$AMJ020200250005
GD20PD20LR20SR30SR31PD30SR40SR41SR42PD40
SR50SR51PD50SR60PD60H
$DTA
*
$END0202
$AMJ030200100002
GD20GD30LR31SR22LR32SR23PD20SR40PD40H
$DTA
CAT CAN
    EAT RAT
$END0302
$AMJ010200080002
GD20LR26CR20BT06GD30PD30PD20H
$DTA
RAM  IS OLDER THAN  SHRIRAM
NOT IN EXISTANCE
$END0102
$AMJ040100120004
GD20PD20GD30PD30GD40GD50LR20CR30BT10PD40
PD50H
$DTA
ABCD
ABCD
DO NOT
MATCH
```

```
$END0401
$AMJ000100050002
GD10PD10GD20PD20H
$DTA
HELLO
This was my complete OS phase 1
$END
```

OS phase 2

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

struct PCB {
    int job_id;
    int TTL;            // total time limit
    int TLL;            // total line limit
    int TTC;            // total time counter
    int TLC;            // total line counter
};

struct PCB proc;

int ptr;                   // page table register
int visited[30];              // virtual group of 10
char M[300][4];                // main memory
char IR[4];                 // instruction register
char R[4];                  // register
int IC;                    // instruction counter
int C;                     // toggle register
int SI;                    // system interrupt
int VA;
int RA;
int PI;                    // program interrupt
int TI;                    // time interrupt
int EM;                      // error message

FILE *inFile;
FILE *outFile;

char *errors[] = {
    "No Error",
    "Out of Data",
    "Line Limit Exceeded",
    "Time Limit Exceeded",
    "Operation Code Error",
    "Operand Error",
    "Invalid Page Fault"
};

void init() {
    for (int i = 0; i < 300; i++) {                 // clearing memory
        for (int j = 0; j < 4; j++) {
            M[i][j] = ' ';
        }
    }

    for (int i = 0; i < 30; i++) {                  // clearing visited flags
        visited[i] = 0;
    }

    for (int i = 0; i < 4; i++) {
        IR[i] = '-';
        R[i] = '-';
    }

    IC = 0;
    C = 0;
    ptr = 0;
    VA = 0;
    PI = 0;
    TI = 0;
    EM = 0;
}

int ALLOCATE() {                    // return a random value lower than 30
    return (rand() % 30);
}

int ADDRESSMAP(int va) {            // error 6
    int pte = ptr * 10 + va / 10;   // page table entry, register virtual address
    char temp[5] = "";

    if (M[pte][0] == '*') {
        printf("Page Fault\n");     // page fault 66
        return -1;
    } else {
        for (int i = 0; i < 4; i++) {
            if (M[pte][i] != ' ')
                strncat(temp, &M[pte][i], 1);
        }
        return ((atoi(temp) * 10) + (va % 10));
    }
}

int terminate(int Code) {          // print in file cause of termination in case of error
    printf("\n%s\n", errors[Code]);
    fprintf(outFile, "\nProgram Terminated abnormally\n%s\n\n", errors[Code]);
    return 0;
}

void MOS() {                                      // errors 1, 2
    if (SI == 1) {
        char line[41];
        if (fgets(line, sizeof(line), inFile) == NULL) {
            EM = 1;
            terminate(1);                       // error due to end when data is asked
            return;
        }
        if (strncmp(line, "$END", 4) == 0) {        // $end
            EM = 1;
            terminate(1);
            return;
        }

        int frame = ALLOCATE();
        while (visited[frame] != 0) {
            frame = ALLOCATE();
        }
        visited[frame] = 1;

        int i = ptr * 10;
        while (M[i][0] != '*') {
            i++;
        }

        int temp = frame / 10;
        M[i][0] = ' ';
        M[i][1] = ' ';
        M[i][2] = temp + '0';
        M[i][3] = frame % 10 + '0';

        int l = 0;
        frame = frame * 10;
        for (int j = 0; j < strlen(line) && strlen(line) < 40; j++) {
            M[frame][l++] = line[j];
            if (l == 4) {
                l = 0;
                frame++;
            }
        }
    } else if (SI == 2) {
        proc.TLC++;                // increase line counter
        if (proc.TLC > proc.TLL) {
            EM = 2;
            terminate(2);                  // line limit exceeded
            return;
        }
        int add = IR[2] - '0';
        add = add * 10;
        int ra = ADDRESSMAP(add);

        if (ra != -1) {
            char out[41] = "";
            for (int i = 0; i < 10; i++) {
                for (int j = 0; j < 4; j++) {
                    strncat(out, &M[ra][j], 1);
                }
                ra++;
            }
            fprintf(outFile, "%s\n", out);
        } else {
            EM = 6;
            terminate(6);                  // invalid cache found
            PI = 3;
        }
    } else if (SI == 3) {
        fprintf(outFile, "\nProgram Terminated successfully\n");
        fprintf(outFile, "IC = %d\tToggle: %d\tTLC: %d\tTTC: %d\tTTL: %d\tTLL: 
%d\tJobId: %d\n",
                IC, C, proc.TLC, proc.TTC, proc.TTL, proc.TLL, proc.job_id);
        for (int i = 0; i < 3; i++) {
            fprintf(outFile, "\t%c", IR[i]);
        }
    }
}

void EXECUTE() {
    while (1) {
        if (PI != 0 || TI != 0 || EM != 0) {
            fprintf(outFile, "IC = %d\tToggle: %d\tTLC: %d\tTTC: %d\tTTL: %d\tTLL:
%d\tJobId:
%d\n",
                    IC, C, proc.TLC, proc.TTC, proc.TTL, proc.TLL, proc.job_id);
            for (int i = 0; i < 3; i++) {
                fprintf(outFile, "\t%c", IR[i]);
            }
            break;
        }

        RA = ADDRESSMAP(IC);
        if (M[RA][0] != 'H' && (!isdigit(M[RA][2]) || !isdigit(M[RA][3]))) {
            EM = 5;
            terminate(5);                   // Operand error
            fprintf(outFile, "IC = %d\tToggle: %d\tTLC: %d\tTTC: %d\tTTL: %d\tTLL:
%d\tJobId:
%d\n",
                    IC, C, proc.TLC, proc.TTC, proc.TTL, proc.TLL, proc.job_id);
            for (int i = 0; i < 3; i++) {
                fprintf(outFile, "\t%c", IR[i]);
            }
        }

        for (int i = 0; i < 4; i++) {
            IR[i] = M[RA][i];
        }
        IC++;

        int add = IR[2] - '0';
        add = add * 10 + (IR[3] - '0');

        if ((IR[0] == 'G' && IR[1] == 'D') || (IR[0] == 'S' && IR[1] == 'R'))
            proc.TTC += 2;
        else
            proc.TTC += 1;

        if (proc.TTC > proc.TTL) {
            EM = 3;
            TI = 2;
            terminate(3);                    // Time limit exceeded
            fprintf(outFile, "IC = %d\tToggle: %d\tTLC: %d\tTTC: %d\tTTL: %d\tTLL:
%d\tJobId:
%d\n",
                    IC, C, proc.TLC, proc.TTC, proc.TTL, proc.TLL, proc.job_id);
            for (int i = 0; i < 3; i++) {
                fprintf(outFile, "\t%c", IR[i]);
            }
            break;
        }
    }
}

void LOAD() {
    printf("\nReading Data...\n");
    char line[41];
    while (fgets(line, sizeof(line), inFile)) {
        if (strncmp(line, "$AMJ", 4) == 0) {
            init();
            ptr = ALLOCATE();
            for (int i = ptr * 10; i < ptr * 10 + 10; i++) {
                for (int j = 0; j < 4; j++) {
                    M[i][j] = '*';
                }
            }
            visited[ptr] = 1;

            char jobid_str[5] = "", TTL_str[5] = "", TLL_str[5] = "";
            strncat(jobid_str, line + 4, 4);
            strncat(TTL_str, line + 8, 4);
            strncat(TLL_str, line + 12, 4);

            proc.job_id = atoi(jobid_str);
            proc.TTL = atoi(TTL_str);
            proc.TLL = atoi(TLL_str);

            printf("IC = %d\tToggle: %d\tTLC: %d\tTTC: %d\tTTL: %d\tTLL: %d\tJobId:
%d\n",
                   IC, C, proc.TLC, proc.TTC, proc.TTL, proc.TLL, proc.job_id);
        } else if (strncmp(line, "$DTA", 4) == 0) {
            EXECUTE();
        } else if (strncmp(line, "$END", 4) == 0) {
            printf("Job ID = %d Ended\n", proc.job_id);
        } else {
            int frame = ALLOCATE();
            while (visited[frame] != 0) {
                frame = ALLOCATE();
            }
            visited[frame] = 1;

            int i = ptr * 10;
            while (M[i][0] != '*') {
                i++;
            }

            int temp = frame / 10;
            M[i][0] = ' ';
            M[i][1] = ' ';
            M[i][2] = temp + '0';
            M[i][3] = frame % 10 + '0';

            int l = 0;
            frame = frame * 10;
            for (int j = 0; j < strlen(line) && j < 40; j++) {
                M[frame][l++] = line[j];
                if (l == 4) {
                    l = 0;
                    frame++;
                }
            }
        }
    }
}

int main() {
    inFile = fopen("input2.txt", "r");
    if (inFile == NULL) {
        perror("Error opening input file");
        return 1;
    }

    outFile = fopen("output2.txt", "w");
    if (outFile == NULL) {
        perror("Error opening output file");
        fclose(inFile);
        return 1;
    }

    LOAD();

    fclose(inFile);
    fclose(outFile);
    return 0;
}
```

Input.txt

```
$AMJ000100050002
GD10PD10H
```

```
$DTA
HELLO-WORLD
$END0001
$AMJ000200060001
GD10LR30SR20PD20H
$DTA
VIT
$END0002
$AMJ000300140003
GD10GD20GD30GD40LR10CO20BT08PD30PD40H
$DTA
2
3
4 -
$END0003
$AMJ000400040001
GD10GD20GD30GD40LR10CR20BT09PD30HPD40H
$DTA
VIT
VIIT
VIT VIIT NOT SAME
VIT VIIT SAME
$END0004
$AMJ000500190001
GD50PD50H
$DTA
4 8 12 16 20 24 28 32 36 40
$END0005
$AMJ000600050001
GD10PD10PD10H
$DTA
HELLO-WORLD
$END0006
$AMJ000700060001
GD10GD20PD10H
$DTA
$END0007
$AMJ000900170007
GD2xPD20LR20SR21PD20SR22PD20SR23PD20SR24
PD20H
$DTA
*
$END0009
```

1. Change your password to a password you would like to use for the remainder of the semester.

```
=> passwd
```

2. Display the system’s date.

```
=> date
```

3. Count the number of lines in the /etc/passwd file.

```
=> wc -l /etc/passwd
```

4. Find out who else is on the system.

```
=> who
```

5. Direct the output of the man pages for the date command to a file name mydate.

```
=> man date > mydate

   ls
```

6. Create a subdirectory called mydir.

```
=> mkdir mydir
```

7. Move the file mydate into the new subdirectory.

```
=> cd/Users/student/mydir

   cp mydate.txt /users/student/pictures

   cd /users/student/pictures

   ls
```

8. Go to the subdirectory mydir and copy the file mydate to a new file called ourdate

```
=> mv mydate mydir
     touch ourdate.txt
     cd mydir
      cp mydate ourdate
```

9. List the contents of mydir.

```
=> cd /users/student/mydir
     ls
```

10. Do a long listing on the file ourdate and note the permissions.

```
=> ls -la
```

11. Display the name of the current directory starting from the root.

```
=> pwd
```

12. Move the files in the directory mydir back to your home directory.

```
=> pwd
     ls
     mv mydate mydate.txt mydir ourdate /users/student
     ls
     cd /users/student
     ls
```

13. Display the first 5 lines of mydate.

```
=> head -5 mydate
```

14. Display the last 8 lines of mydate.

```
=> tail -8 mydate
```

15. Remove the directory mydir.

```
=> rm -r mydir

   ls
```

16. Redirect the output of the long listing of files to a file named list.

```
=> ls

   ls > list

   cat list
```

17. Select any 5 capitals of states in India and enter them in a file named capitals1. Choose 5 more capitals and enter them in a file named capitals2. Choose 5 more capitals and enter them in a file named capitals3. Concatenate all 3 files and redirect the output to a file named capitals.

```
=> echo “c1 c2 c3 c3 c4 c5” > capitals1

  echo “c6 c7 c8 c9 c10” > capitals2

  echo “c11 c12 c13  c14 c15” > capitals3

  cat capitals1 capitals2 capitals3 > capitals
```

18. Concatenate the file capitals2 at the end of file capitals.

```
=> cat capitals capitals2
```

19. Give read and write permissions to all users for the file capitals.

```
=> chmod a+rw capitals

   ls -l capitals
```

20. Give read permissions only to the owner of the file capitals. Open the file, make some changes and try to save it. What happens ?

```
=> Error Writing capitals : Permissions denied
```

21. the output to a file named capitals. Activate the alias and make it run.

```
=>  alias concat3in1 = “cat capitals capitals2 capitals3 > capitals”

    concat3in1

    ls
```

22. Find out the number of times the string “the” appears in the file mydate

```
=> grep -c “the” mydate
```

23. Find out the lines numbers on which the string “date” exists in mydate

```
=> grep -n “date” mydate
```

24. Print all lines of mydate except those that have the letter “l” in them

```
=> grep -v “l” mydate
```

25. List the words of 4 letters from the file mydate

```
=> grep -o -w “\w\{4\}” mydate
```

26. List 5 states in north east India in a file mystates . List their corresponding capitals in a file

Mycapitals Use the paste command to join the 2 files

```
    => nano mystates

       nano mycapitals

       paste mystates mycapitals
```

27. Use the cut command to print the 1st and 3rd columns of the /etc /passwd file for all students in the class.

```
=> cut -c 1,3 /etc/passwd
```

28. Count the number of people logged in and also trap the users in a file using the tee Command.

```
=> who | tee users.txt | wc -l
```

29. Convert the contents of mystates into uppercase.

```
=> tr a-z A-Z<mystates
```

30. Create any two files & display the common values between them.

```
=> touch abc.txt
   touch xyz.txt
   echo “a \nb \nc \nx \nt \ny \nz” > xyz.txt
   echo “l  \nk \ng \nf \ny \nz” > abc.txt
   comm -12 <(sort abc.txt) < (sort xyz.txt)
```