ASSIGNEMNT 2

Write a script to print the date.

```
#!/bin/bash

# Print the current date
echo "Today's date is: $(date)"
```

Write a script to display the name of user received.

```
#!/bin/bash

# Check if a username is provided
if [ -z "$1" ]; then
    echo "Usage: $0 <username>"
    exit 1
fi

# Display the provided username
echo "Hello, $1!"

chmod +x display_user.sh
./display_user.sh John
```

3. Write a script to find the greatest of three numbers (numbers passed as command line parameters).

```
#!/bin/bash

# Check if exactly three numbers are provided
if [ $# -ne 3 ]; then
    echo "Usage: $0 <num1> <num2> <num3>"
    exit 1
fi

# Compare the numbers and find the greatest
if [ $1 -gt $2 ] && [ $1 -gt $3 ]; then
    echo "$1 is the greatest number."
elif [ $2 -gt $1 ] && [ $2 -gt $3 ]; then
    echo "$2 is the greatest number."
else
    echo "$3 is the greatest number."
Fi

chmod +x greatest_of_three.sh
./greatest_of_three.sh 12 45 23
```

Write a script to check whether the given no. is even/odd.

```
#!/bin/bash

# Check if a number is provided
if [ -z "$1" ]; then
    echo "Usage: $0 <number>"
    exit 1
fi

# Check if the number is even or odd
if [ $(($1 % 2)) -eq 0 ]; then
    echo "$1 is even."
else
    echo "$1 is odd."
Fi

chmod +x check_even_odd.sh
./check_even_odd.sh 5
```

ASSIGNEMNT 3

1. Write a script to calculate the average of n numbers.

```
#!/bin/bash
# Script to calculate the average of n numbers
echo " Enter numbers separated by space:"
read -a numbers
sum=0
count=${#numbers[@]}
for num in "${numbers[@]}"
do
sum=$(echo "$sum + $num" | bc)
done
average=$(echo "scale=2; $sum / $count" | bc)
echo "Average of the numbers is: $average"
```

2. Write a script to check whether the given number is prime or not.

```
#!/bin/bash
# Script to check whether the given number is prime or not
echo “Enter a number:”
read number
if [ $number -le 1 ]; then
echo “$number is not a prime number”
exit 0
fi
is_prime=1
for (( i=2; i*i<==number; i++ ))
do
if [ $(($number % $i)) -eq 0 ]; then
is_prime=0
break
fi
done
if [ $is_prime -eq 1 ]; then
echo “$number is a prime number.”
else
echo “$number is not a prime number.”
fi
```

3. Write a script to check whether the given input is a number or a string.

```
#!/bin/bash
# Script to check whether the given input is a number or a string
echo “Enter something:”
read input
if [[ “$input” =~ ^[+-]?[0-9]+\.?[0-9]*$ ]]; then
echo “The input &amp;#39;$input&amp;#39; is a Number”
else
echo “The input &amp;#39;$input&amp;#39; is a String.”
Fi
```

ASSIGNMENT 4

Write a script to print the Fibonacci series up to n terms.

```
#!/bin/bash

read -p "Enter the number of terms: " n
a=0
b=1
echo -n "Fibonacci series: "
for (( i=1; i<=n; i++ ))
do
   echo -n "$a "
   temp=$((a + b))
   a=$b
   b=$temp
done
echo
```

2. Write a script to calculate the factorial of a given number.

```
#!/bin/bash

read -p "Enter a number: " n
fact=1
for (( i=1; i<=n; i++ ))
do
   fact=$((fact * i))
done
echo "Factorial of $n is $fact"
```

Write a script to calculate the sum of digits of the given number.

```
#!/bin/bash

read -p "Enter a number: " num
sum=0

# Loop to sum the digits
while [ $num -gt 0 ]
do
   digit=$((num % 10))
   sum=$((sum + digit))
   num=$((num / 10))
done

echo "Sum of digits is $sum"
```

ASSIGNEMNT 5

Write a script to compute no. of characters and words in each line .

```
#!/bin/bash
echo “Enter lines of text (Ctrl+D to end):”
while IFS= read -r line; do
char_count=$(echo -n “$line” | wc -c)
word_count=$(echo -n “$line” | wc -w)
echo “Line: $line”
echo “Characters: $char_count, Words: $word_count”
echo “----------"
done
```

Write a script to check whether the given string is a palindrome

```
#!/bin/bash

read -p "Enter a string: " str
rev_str=$(echo "$str" | rev)

if [[ "$str" == "$rev_str" ]]; then
   echo "$str is a palindrome."
else
   echo "$str is not a palindrome."
Fi
```

Write a shell script that accepts a string from the terminal and echo a suitable message if it doesn't have at least 5 characters including the other symbols.

```
#!/bin/bash

read -p "Enter a string: " str

if [ ${#str} -ge 5 ]; then
   echo "String is valid."
else
   echo "String must have at least 5 characters."
Fi
```

Write a shell script to echo the string length of the given string as argument.

```
echo “Enter a string:”
read string
echo “The length of the string is ${#string}.”
```

ASSIGNEMNT 6

. Write a shell script that accepts two directory names as arguments and deletes those files in the first directory which are similarly named in the second directly. Note: Contents should also match inside the files.

```
#!/bin/bash

# Check if two arguments are provided
if [ $# -ne 2 ]; then
   echo "Usage: $0 <dir1> <dir2>"
   exit 1
fi

dir1=$1
dir2=$2

# Loop through files in the first directory
for file in "$dir1"/*; do
   filename=$(basename "$file")
   file2="$dir2/$filename"

   # Check if file exists in both directories and contents match
   if [ -f "$file2" ] && cmp -s "$file" "$file2"; then
      rm "$file"
      echo "Deleted $file"
   fi
done
#### create 2 directoryu
mkdir dir1
mkdir dir2

cd ~/Documents
mkdir dir1 dir2
```

Write a shell script to display the processes running on the system for every 30 seconds, but only for 3 times.

```
#!/bin/bash
iterations=3
delay=30
for i in $(seq 1 $iterations); do
echo &amp;quot;Iteration $i: Displaying processes&amp;quot;
ps aux
[ $i -lt $iterations ] &amp;amp;&amp;amp; sleep $delay
done
echo &amp;quot;Process display completed $iterations times.&amp;quot;
```

Write a shell script that displays the last modification time of any file.

```
#!/bin/bash

# Check if a file name is provided
if [ -z "$1" ]; then
   echo "Usage: $0 <filename>"
   exit 1
fi

# Display the last modification time
mod_time=$(stat -f "%Sm" -t "%Y-%m-%d %H:%M:%S" "$1")
echo "Last modification time of $1 is: $mod_time"
```

```
chmod +x mod_time.sh
./mod_time.sh <file_name>
```

Create a file name first and then give name while running

Write a shell script to check the spellings of any text document given as an argument.

```
#!/bin/bash

# Check if a file name is provided
if [ -z "$1" ]; then
   echo "Usage: $0 <filename>"
   exit 1
fi

# Check if the file exists
if [ ! -f "$1" ]; then
   echo "File not found!"
   exit 1
fi

# Check spelling using aspell
aspell check "$1"
```

```
chmod +x spell_check.sh
./spell_check.sh yourfile.txt
./spell_check.sh document.txt
brew install aspell
sudo apt-get install aspell
```

ASSIGNEMNT 7

1. Write a CPU scheduling algorithms using C programming.
   • First Come First Serve
   • Shortest Job First
   • Priority
   • Round Robin

First Come First Serve (FCFS)

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

```
void FCFS(int processes[], int n, int burst[]) {
    int wait[n], turnaround[n], total_wait = 0, total_turnaround = 0;

    wait[0] = 0; // Waiting time for the first process is 0

    for (int i = 1; i < n; i++) {
        wait[i] = wait[i - 1] + burst[i - 1];
    }

    for (int i = 0; i < n; i++) {
        turnaround[i] = wait[i] + burst[i];
        total_wait += wait[i];
        total_turnaround += turnaround[i];
    }

    printf("Process\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], burst[i], wait[i], turnaround[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)total_wait / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround / n);
}

int main() {
    int processes[] = {1, 2, 3};
    int n = sizeof(processes) / sizeof(processes[0]);
    int burst[] = {5, 8, 12};

    FCFS(processes, n, burst);
    return 0;
}
```

Shortest Job First (SJF)

```
#include <stdio.h>

void SJF(int processes[], int n, int burst[]) {
    int wait[n], turnaround[n], total_wait = 0, total_turnaround = 0;
    int temp;

    // Sort burst times and processes
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
```

```
            if (burst[i] > burst[j]) {
                temp = burst[i];
                burst[i] = burst[j];
                burst[j] = temp;

                temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }

    wait[0] = 0;

    for (int i = 1; i < n; i++) {
        wait[i] = wait[i - 1] + burst[i - 1];
    }

    for (int i = 0; i < n; i++) {
        turnaround[i] = wait[i] + burst[i];
        total_wait += wait[i];
        total_turnaround += turnaround[i];
    }

    printf("Process\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], burst[i], wait[i], turnaround[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)total_wait / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround / n);
}

int main() {
    int processes[] = {1, 2, 3};
    int n = sizeof(processes) / sizeof(processes[0]);
    int burst[] = {6, 8, 7};

    SJF(processes, n, burst);
    return 0;
}
```

. Priority Scheduling

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

```
void Priority(int processes[], int n, int burst[], int priority[]) {
   int wait[n], turnaround[n], total_wait = 0, total_turnaround = 0, temp;

   // Sort processes based on priority
   for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
         if (priority[i] > priority[j]) {
            temp = priority[i];
            priority[i] = priority[j];
            priority[j] = temp;

            temp = burst[i];
            burst[i] = burst[j];
            burst[j] = temp;

            temp = processes[i];
            processes[i] = processes[j];
            processes[j] = temp;
         }
      }
   }

   wait[0] = 0;

   for (int i = 1; i < n; i++) {
      wait[i] = wait[i - 1] + burst[i - 1];
   }

   for (int i = 0; i < n; i++) {
      turnaround[i] = wait[i] + burst[i];
      total_wait += wait[i];
      total_turnaround += turnaround[i];
   }

   printf("Process\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
   for (int i = 0; i < n; i++) {
      printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], burst[i], priority[i], wait[i],
turnaround[i]);
   }

   printf("\nAverage Waiting Time: %.2f\n", (float)total_wait / n);
   printf("Average Turnaround Time: %.2f\n", (float)total_turnaround / n);
}

int main() {
   int processes[] = {1, 2, 3};
```

```
    int n = sizeof(processes) / sizeof(processes[0]);
    int burst[] = {10, 5, 8};
    int priority[] = {2, 1, 3};

    Priority(processes, n, burst, priority);
    return 0;
}
```

Round Robin Scheduling

```
#include <stdio.h>

void RoundRobin(int processes[], int n, int burst[], int quantum) {
    int remaining[n], wait[n], turnaround[n], total_wait = 0, total_turnaround = 0, time = 0;

    for (int i = 0; i < n; i++) {
        remaining[i] = burst[i];
        wait[i] = 0;
    }

    while (1) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (remaining[i] > 0) {
                done = 0;
                if (remaining[i] > quantum) {
                    time += quantum;
                    remaining[i] -= quantum;
                } else {
                    time += remaining[i];
                    wait[i] = time - burst[i];
                    remaining[i] = 0;
                }
            }
        }
        if (done) break;
    }

    for (int i = 0; i < n; i++) {
        turnaround[i] = burst[i] + wait[i];
        total_wait += wait[i];
        total_turnaround += turnaround[i];
    }

    printf("Process\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], burst[i], wait[i], turnaround[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)total_wait / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround / n);
}

int main() {
    int processes[] = {1, 2, 3};
    int n = sizeof(processes) / sizeof(processes[0]);
    int burst[] = {24, 3, 3};
    int quantum = 4;

    RoundRobin(processes, n, burst, quantum);
    return 0;
}
```

ASSIGNMENT 8

Write a program using C for producer consumer problem using shared memory method.

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

#define BUFFER_SIZE 5

struct shared_memory {
    int buffer[BUFFER_SIZE];
    int in, out;
};

void producer(int shmid, int semid) {
    struct shared_memory *shm = (struct shared_memory *)shmat(shmid, NULL, 0);
    int item = 0;

    for (int i = 0; i < 10; i++) {
        struct sembuf wait_empty = {0, -1, 0}; // Wait for empty slot
        struct sembuf wait_mutex = {1, -1, 0}; // Wait for mutex
        struct sembuf signal_mutex = {1, 1, 0}; // Signal mutex
        struct sembuf signal_full = {2, 1, 0}; // Signal full slot

        item = rand() % 100;
```

```
        semop(semid, &wait_empty, 1);
        semop(semid, &wait_mutex, 1);

        shm->buffer[shm->in] = item;
        printf("Produced: %d\n", item);
        shm->in = (shm->in + 1) % BUFFER_SIZE;

        semop(semid, &signal_mutex, 1);
        semop(semid, &signal_full, 1);

        sleep(1);
    }
}

void consumer(int shmid, int semid) {
    struct shared_memory *shm = (struct shared_memory *)shmat(shmid, NULL, 0);

    for (int i = 0; i < 10; i++) {
        struct sembuf wait_full = {2, -1, 0}; // Wait for full slot
        struct sembuf wait_mutex = {1, -1, 0}; // Wait for mutex
        struct sembuf signal_mutex = {1, 1, 0}; // Signal mutex
        struct sembuf signal_empty = {0, 1, 0}; // Signal empty slot

        semop(semid, &wait_full, 1);
        semop(semid, &wait_mutex, 1);

        int item = shm->buffer[shm->out];
        printf("Consumed: %d\n", item);
        shm->out = (shm->out + 1) % BUFFER_SIZE;

        semop(semid, &signal_mutex, 1);
        semop(semid, &signal_empty, 1);

        sleep(2);
    }
}

int main() {
    int shmid = shmget(IPC_PRIVATE, sizeof(struct shared_memory), IPC_CREAT | 0666);
    int semid = semget(IPC_PRIVATE, 3, IPC_CREAT | 0666);

    semctl(semid, 0, SETVAL, BUFFER_SIZE); // Empty slots
    semctl(semid, 1, SETVAL, 1);         // Mutex
    semctl(semid, 2, SETVAL, 0);         // Full slots

    if (fork() == 0) {
        producer(shmid, semid);
```

```
        exit(0);
    } else {
        consumer(shmid, semid);
        wait(NULL);
    }

    shmctl(shmid, IPC_RMID, NULL);
    semctl(semid, 0, IPC_RMID);
    return 0;
}
```

Producer-Consumer Problem Using Message Passing

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

struct message {
    long msg_type;
    int data;
};

void producer(int msgid) {
    struct message msg;
    msg.msg_type = 1;

    for (int i = 0; i < 10; i++) {
        msg.data = rand() % 100;
        msgsnd(msgid, &msg, sizeof(msg.data), 0);
        printf("Produced: %d\n", msg.data);
        sleep(1);
    }
}

void consumer(int msgid) {
    struct message msg;

    for (int i = 0; i < 10; i++) {
        msgrcv(msgid, &msg, sizeof(msg.data), 1, 0);
        printf("Consumed: %d\n", msg.data);
        sleep(2);
    }
}

int main() {
```

```
    int msgid = msgget(IPC_PRIVATE, IPC_CREAT | 0666);

    if (fork() == 0) {
        producer(msgid);
        exit(0);
    } else {
        consumer(msgid);
        wait(NULL);
    }

    msgctl(msgid, IPC_RMID, NULL);
    return 0;
}
```

Program with Two Threads (Hello and World)

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

void *print_hello(void *arg) {
    printf("Hello\n");
    return NULL;
}

void *print_world(void *arg) {
    printf("World\n");
    return NULL;
}

int main() {
    pthread_t thread1, thread2;

    pthread_create(&thread1, NULL, print_hello, NULL);
    pthread_create(&thread2, NULL, print_world, NULL);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;
}
```

Compilation and Execution:

1. Save each program in a `.c` file.

2. Compile using:

```
gcc -o program_name program_name.c -lpthread
```

3. Run the compiled program:

```
./program_name
```

```
ASSIGNMENT 9
```

1. Write a shell script which reads the contents in a text file and removes all the blank spaces in them and redirects the output to a file

```
#!/bin/bash
# Remove spaces from the text file and redirect output to another file

if [ $# -ne 2 ]; then
    echo "Usage: $0 input_file output_file"
    exit 1
fi

tr -d ' ' < "$1" > "$2"
echo "Spaces removed and output redirected to $2"
```

2. Write a shell script that changes the name of the files passed as arguments to lowercase.

```
#!/bin/bash
for file in "$@"; do
       lower_file=$(echo "$file" | tr 'A-Z' 'a-z')
        if [ "$file" != "$lower_file" ]; then
        mv "$file" "$lower_file"
        echo "Renamed $file to $lower_file"
    else
        echo "$file is already in lowercase"
    fi
done
```

3.. Write a shell script to translate all the characters to lower case in a given text file.

```
#!/bin/bash
# Translate characters to lowercase in a text file

if [ $# -ne 2 ]; then
    echo "Usage: $0 input_file output_file"
    exit 1
fi

tr '[:upper:]' '[:lower:]' < "$1" > "$2"
echo "Translated to lowercase and saved to $2"
```

4. Write a shell script to combine any three text files into a single file (append them in the order as they appear in the arguments) and display the word count.

```
#!/bin/bash
# Combine three text files and display the word count

if [ $# -ne 3 ]; then
   echo "Usage: $0 file1 file2 file3"
   exit 1
fi

cat "$1" "$2" "$3" > combined.txt
echo "Combined contents into combined.txt"
wc -w combined.txt
```

Assignment 9

Shell Script to Separate Even and Odd Numbered Lines

```
bash
Copy code
#!/bin/bash

if [ -z "$1" ]; then
    echo "Usage: $0 <filename>"
    exit 1
fi

awk 'NR % 2 == 0 { print > "evenfile" } NR % 2 == 1 { print > "oddfile" }'
"$1"

echo "Even lines written to evenfile and odd lines written to oddfile."
```

---

2. Shell Script to Delete All Even Numbered Lines

```
bash
Copy code
#!/bin/bash

if [ -z "$1" ]; then
    echo "Usage: $0 <filename>"
    exit 1
fi

awk 'NR % 2 == 1' "$1" > temp && mv temp "$1"

echo "Even-numbered lines deleted from $1."
```

---

3. Shell Script: User Info with Asterisks

```
bash
Copy code
#!/bin/bash

echo "Username: $USER"
echo "**********"
echo "Date and Time: $(date)"
echo "**********"
echo "Users Logged On:"
who
echo "**********"
```

---

4. Shell Script to Count Files in Subdirectories

```
bash
Copy code
#!/bin/bash

for dir in */; do
    if [ -d "$dir" ]; then
        count=$(find "$dir" -type f | wc -l)
        echo "$dir: $count files"
    fi
done
```

---

5. C Program to Simulate Banker's Algorithm

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

void calculateNeed(int need[][10], int max[][10], int allocation[][10], int
processes, int resources) {
    for (int i = 0; i < processes; i++) {
        for (int j = 0; j < resources; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
}

int isSafe(int processes, int resources, int available[], int max[][10],
int allocation[][10]) {
    int need[10][10], finish[10] = {0}, safeSequence[10];
    int work[10];

    calculateNeed(need, max, allocation, processes, resources);

    for (int i = 0; i < resources; i++) {
        work[i] = available[i];
    }

    int count = 0;
    while (count < processes) {
        int found = 0;
        for (int p = 0; p < processes; p++) {
            if (finish[p] == 0) {
                int j;
```

```
                for (j = 0; j < resources; j++) {
                    if (need[p][j] > work[j]) {
                        break;
                    }
                }
                if (j == resources) {
                    for (int k = 0; k < resources; k++) {
                        work[k] += allocation[p][k];
                    }
                    safeSequence[count++] = p;
                    finish[p] = 1;
                    found = 1;
                }
            }
        }
        if (!found) {
            printf("The system is not in a safe state.\n");
            return 0;
        }
    }

    printf("The system is in a safe state.\nSafe Sequence: ");
    for (int i = 0; i < processes; i++) {
        printf("P%d ", safeSequence[i]);
    }
    printf("\n");
    return 1;
}

int main() {
    int processes, resources;
    int max[10][10], allocation[10][10], available[10];

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

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

    printf("Enter the Max Matrix:\n");
    for (int i = 0; i < processes; i++) {
        for (int j = 0; j < resources; j++) {
            scanf("%d", &max[i][j]);
        }
    }

    printf("Enter the Allocation Matrix:\n");
    for (int i = 0; i < processes; i++) {
        for (int j = 0; j < resources; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }

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

    isSafe(processes, resources, available, max, allocation);

    return 0;
```

```
}
```

Compilation and Execution:

1. Save the C program as `bankers.c`.
2. Compile:

   ```
   bash
   Copy code
   gcc -o bankers bankers.c
   ```

3. Run the program:

   ```
   bash
   Copy code
   ./bankers
   ```

Let me know if you need further clarifications!

```
Assignment 11
```

1. Shell Script to Implement a "more" Command

This script displays 15 lines of a file at a time and waits for user input before showing the next set.

```
bash
Copy code
#!/bin/bash

read -p "Enter your name: " username
read -p "Enter the filename: " filename

if [ ! -f "$filename" ]; then
    echo "File does not exist."
    exit 1
fi

lines=15
total_lines=$(wc -l < "$filename")
current_line=1

echo "Hello, $username! Showing the content of $filename in chunks of
$lines lines:"
while [ $current_line -le $total_lines ]; do
    sed -n "${current_line},$((current_line + lines - 1))p" "$filename"
    current_line=$((current_line + lines))
    read -n 1 -s -r -p "Press any key to continue..."
    echo
done
```

---

2. Shell Script to Count English Articles (a, an, the)

This script counts occurrences of "a", "an", and "the" in a given text file.

```
bash
Copy code
#!/bin/bash

if [ -z "$1" ]; then
    echo "Usage: $0 <filename>"
    exit 1
fi

if [ ! -f "$1" ]; then
    echo "File does not exist."
    exit 1
fi

a_count=$(grep -o -i "\ba\b" "$1" | wc -l)
an_count=$(grep -o -i "\ban\b" "$1" | wc -l)
the_count=$(grep -o -i "\bthe\b" "$1" | wc -l)

echo "Occurrences of 'a': $a_count"
echo "Occurrences of 'an': $an_count"
echo "Occurrences of 'the': $the_count"
total=$((a_count + an_count + the_count))
echo "Total English articles: $total"
```

---

3. Shell Script to Replace `c` with `chr` in a String

This script replaces all occurrences of a character `c` with a string `chr` and counts the replacements.

```
bash
Copy code
#!/bin/bash

read -p "Enter the string: " s
read -p "Enter the character to replace: " c
read -p "Enter the replacement string: " chr

modified=$(echo "$s" | sed "s/$c/$chr/g")
count=$(echo "$s" | grep -o "$c" | wc -l)

echo "Modified string: $modified"
echo "Number of replacements: $count"
```

---

4. Shell Script to Concatenate Two Strings

This script concatenates two input strings and displays the result along with its length.

```
bash
```

```
Copy code
#!/bin/bash

read -p "Enter the first string: " str1
read -p "Enter the second string: " str2

result="$str1$str2"
length=${#result}

echo "Concatenated string: $result"
echo "Length of the resultant string: $length"
```

How to Run the Scripts

1. Save the script in a file, e.g., `script.sh`.
2. Make it executable:

   ```
   bash
   Copy code
   chmod +x script.sh
   ```

3. Run the script:

   ```
   bash
   Copy code
   ./script.sh
   ```

Let me know if you need further help!

4o

```
Assignment 12
```

Write a program in C for following Page Replacement Algorithms
• First In First Out
• Least Recently Used
• Optimal
• Least Frequently Used
• Second Chance

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

// Function to check if a page is in a frame
int isInFrame(int frames[], int n, int page) {
    for (int i = 0; i < n; i++) {
        if (frames[i] == page)
            return i;
    }
```

```
    return -1;
}

// FIFO Algorithm
void fifo(int pages[], int n, int capacity) {
    int frames[capacity], front = 0, faults = 0;
    for (int i = 0; i < capacity; i++) frames[i] = -1;

    for (int i = 0; i < n; i++) {
        if (isInFrame(frames, capacity, pages[i]) == -1) {
            frames[front] = pages[i];
            front = (front + 1) % capacity;
            faults++;
        }
    }
    printf("FIFO: Page Faults = %d\n", faults);
}

// LRU Algorithm
void lru(int pages[], int n, int capacity) {
    int frames[capacity], age[capacity], faults = 0;
    for (int i = 0; i < capacity; i++) frames[i] = -1;

    for (int i = 0; i < n; i++) {
        int idx = isInFrame(frames, capacity, pages[i]);
        if (idx == -1) {
            int lruIdx = 0;
            for (int j = 1; j < capacity; j++) {
                if (age[j] < age[lruIdx]) lruIdx = j;
            }
            frames[lruIdx] = pages[i];
            faults++;
            idx = lruIdx;
        }
        age[idx] = i;
    }
    printf("LRU: Page Faults = %d\n", faults);
}

// Optimal Algorithm
void optimal(int pages[], int n, int capacity) {
    int frames[capacity], faults = 0;
    for (int i = 0; i < capacity; i++) frames[i] = -1;

    for (int i = 0; i < n; i++) {
        if (isInFrame(frames, capacity, pages[i]) == -1) {
            int replaceIdx = -1, farthest = -1;
            for (int j = 0; j < capacity; j++) {
                if (frames[j] == -1) {
                    replaceIdx = j;
                    break;
                }
                int nextUse = INT_MAX;
                for (int k = i + 1; k < n; k++) {
```

```
                if (pages[k] == frames[j]) {
                    nextUse = k;
                    break;
                }
            }
            if (nextUse > farthest) {
                farthest = nextUse;
                replaceIdx = j;
            }
        }
        frames[replaceIdx] = pages[i];
        faults++;
    }
  }
  printf("Optimal: Page Faults = %d\n", faults);
}

// LFU Algorithm
void lfu(int pages[], int n, int capacity) {
  int frames[capacity], freq[capacity], faults = 0;
  for (int i = 0; i < capacity; i++) {
    frames[i] = -1;
    freq[i] = 0;
  }

  for (int i = 0; i < n; i++) {
    int idx = isInFrame(frames, capacity, pages[i]);
    if (idx == -1) {
      int lfuIdx = 0;
      for (int j = 1; j < capacity; j++) {
        if (freq[j] < freq[lfuIdx] || (freq[j] == freq[lfuIdx] && frames[j] == -1))
          lfuIdx = j;
      }
      frames[lfuIdx] = pages[i];
      freq[lfuIdx] = 1;
      faults++;
    } else {
      freq[idx]++;
    }
  }
  printf("LFU: Page Faults = %d\n", faults);
}

// Second Chance Algorithm
void secondChance(int pages[], int n, int capacity) {
  int frames[capacity], reference[capacity], front = 0, faults = 0;
  for (int i = 0; i < capacity; i++) {
    frames[i] = -1;
    reference[i] = 0;
  }

  for (int i = 0; i < n; i++) {
    int idx = isInFrame(frames, capacity, pages[i]);
    if (idx == -1) {
```

```
            while (reference[front]) {
                reference[front] = 0;
                front = (front + 1) % capacity;
            }
            frames[front] = pages[i];
            reference[front] = 1;
            front = (front + 1) % capacity;
            faults++;
        } else {
            reference[idx] = 1;
        }
    }
    printf("Second Chance: Page Faults = %d\n", faults);
}

int main() {
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int n = sizeof(pages) / sizeof(pages[0]);
    int capacity = 4;

    fifo(pages, n, capacity);
    lru(pages, n, capacity);
    optimal(pages, n, capacity);
    lfu(pages, n, capacity);
    secondChance(pages, n, capacity);

    return 0;
}
```