# LIST OF EXPERIMENTS

1. Study of basic LINUX & Vi Editor command

2. string and numerical handling functions.

    a) Write a shell script to compare two string.

    b) Write a shell script to extract the first and last character from a string.

    b) Write a shell script to find whether the given number is palindrome or not.

3. Loop and Selection constructs

    a) Write a shell script to find the factorial of a given number using for loop.

    b) Write a shell script to find the sum of n numbers using while loop.

    c) Write a shell script to implement menu driven program to perform all arithmetic operation using case statement.

    d) Write a shell script to print following pattern.

```
*
* *
* * *
* * * *
```

4. File Handling Functions:

    a) Converting File names from Uppercase to Lowercase

    b) Write a shell script to count the number of characters, words and lines in the file.

    c) Write a shell script to read and check the file exists or not, if not create the file.

5. Manipulate Date/Time/Calendar

6. Showing various system information

7. Implementation of process scheduling mechanism – FCFS, SJF, Priority Queue.

8. Reader – Writer Problem.

9. Dinner’s Philosopher Problem.

10. First Fit, Worst Fit, Best Fit allocation strategy.

11. Bankers Algorithm

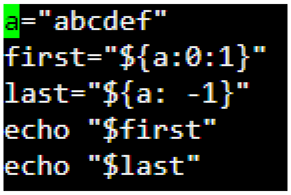
12. Implement the producer consumer problem using Semaphore

13. Implement some memory management Scheme

**EXP. No. 2.a Write a shell script to compare two string.**

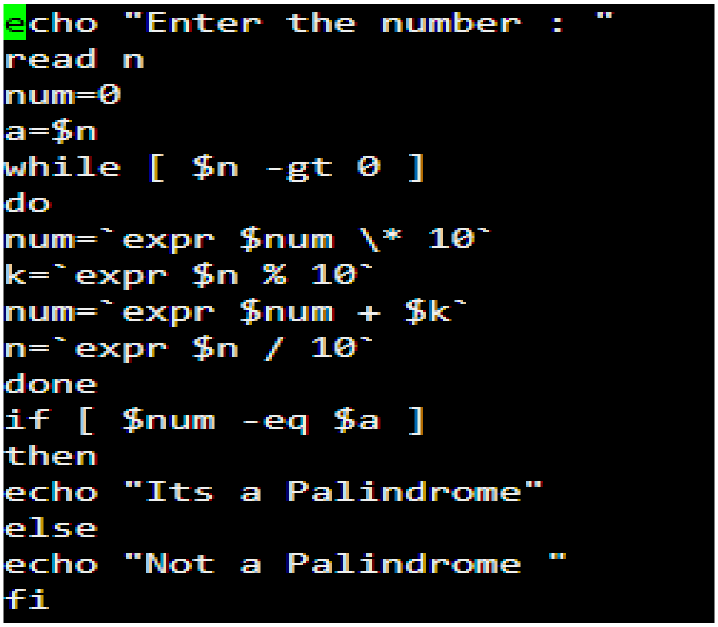
```
echo “Enter two string”
read a
read b
if [ -z $a ]
then
echo “ First String is empty: Null String”
fi
if [ -z $b ]
then
echo “ Second String is empty: Null String”
fi
if [ $a = $b ]
then
echo “Strings are equal: strings Matched”
else
echo “Strings are not equal: Strings not match”
fi
```

**EXP. No. 2.b Write a shell script to extract the first and last character from a string.**



```
a="abcdef"
first="${a:0:1}"
last="${a: -1}"
echo "$first"
echo "$last"
```

**EXP. No. 2.c Write a shell script to check whether the number is palindrome or not.**



```
echo "Enter the number : "
read n
num=0
a=$n
while [ $n -gt 0 ]
do
num=`expr $num \* 10`
k=`expr $n % 10`
num=`expr $num + $k`
n=`expr $n / 10`
done
if [ $num -eq $a ]
then
echo "Its a Palindrome"
else
echo "Not a Palindrome "
fi
```

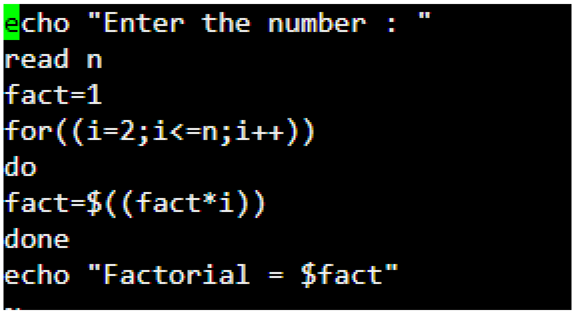
**EXP. No 3.a write a shell script to find the factorial of a given number.**

1. Get a number

2. Use for loop or while loop to compute the factorial by using the below formula

3. fact(n) = n * n-1 * n-2 * .. 1

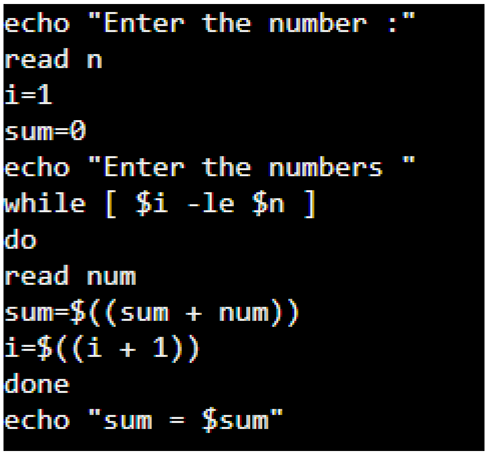
4. Display the result.



```
echo "Enter the number : "
read n
fact=1
for((i=2;i<=n;i++))
do
fact=$((fact*i))
done
echo "Factorial = $fact"
```

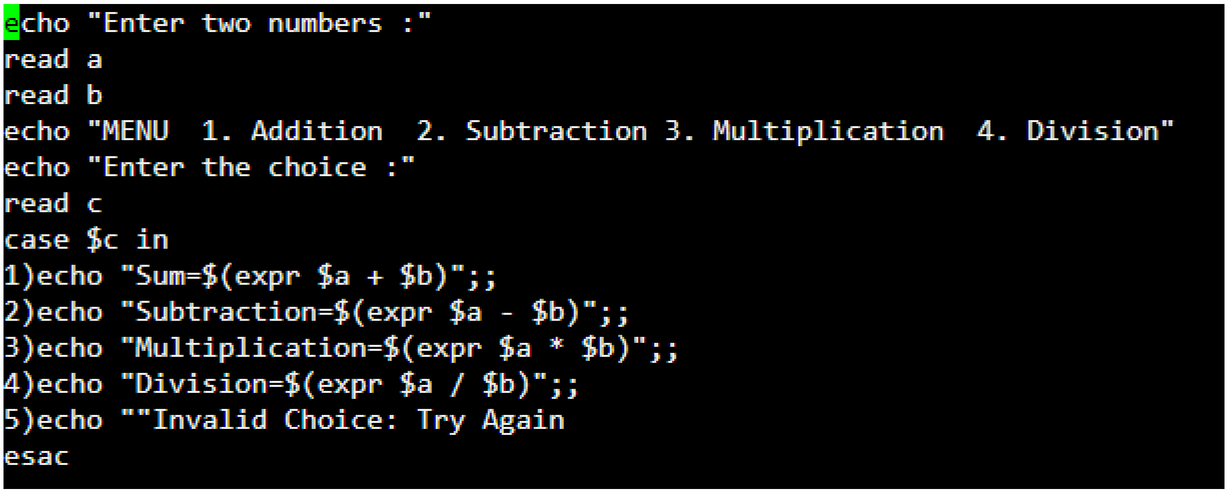
EXP. No 3.b Write a shell script to find the sum of n numbers using while loop.

1. Get N (Total Numbers).
2. Get N numbers using loop.
3. Calculate the sum.
4. Print the result.



```
echo "Enter the number :"
read n
i=1
sum=0
echo "Enter the numbers "
while [ $i -le $n ]
do
read num
sum=$((sum + num))
i=$((i + 1))
done
echo "sum = $sum"
```

EXP. No 3.c Write a shell script to implement menu driven program to perform all arithmetic operation using case statement.



```
echo "Enter two numbers :"
read a
read b
echo "MENU  1. Addition  2. Subtraction 3. Multiplication  4. Division"
echo "Enter the choice :"
read c
case $c in
1)echo "Sum=$(expr $a + $b)";;
2)echo "Subtraction=$(expr $a - $b)";;
3)echo "Multiplication=$(expr $a * $b)";;
4)echo "Division=$(expr $a / $b)";;
5)echo ""Invalid Choice: Try Again
esac
```

### EXP. No 3.d Write a shell script to print following pattern.

```
*
* *
* * *
* * * *
```

```
echo “ enter number of rows”
read n
i=1
while [  $i –le $n ]
do
j=1
while [ $j –le $i ]
do
echo –n “*”
j=$((j + 1))
done
echo
i=$((i + 1))
done
```

### EXP. No. 4.a Converting File names from Uppercase to Lowercase

```
var_name="THIS IS a TEST"
echo "$VAR_NAME" | tr '[:upper:]' '[:lower:]'
name="sathyabama"
echo "$name" | tr '[:lower:]' '[:upper:]'
```

### EXP. No. 4.b Write a shell script to count the number of characters, words and lines in the file.

**Create a file :**

```
cat > filename
My university name is Sathyabama
My  Department is Computer Science
This is Chennai
```

**Program:**

```
echo "enter file name"
read file
c=`cat $file | wc -c`
w=`cat $file | wc -w`
l=`grep -c "." $file`
echo "Number of characters is $c"
echo "Number of words is $w"
echo "Number of Lines is $l"
```

**EXP. No. 4.c Write a shell script to read and check the file exists or not, if not create the file.**

```
echo “enter name of file”
read filename
if [ -f $filename ]
then
echo “File $filename Exits!”
else
touch $filename
fi
```

EXP. No. 5 Write a shell script to Manipulate Date/Time/Calendar.

```
echo "Date in various forms"
echo $(date)
echo "Today is $(date +'%m/%d/%y')"
echo "Today is $(date +'%Y-%m-%d')"
echo "Calender is various form"
echo $(cal 9 2024)
echo $(cal 2024)
echo $(cal -m May)
echo "Time in various formats"
echo $(date +"%T")
echo $(date +"%r")
echo $(date +"%I:%S:%M")
```

EXP. No. 6 Write a shell script Showing various system information

```
echo "SYSTEM INFORMATION"
echo “Hello ,$LOGNAME”
echo “Current Date is = $(date)”
echo “User is ‘who I am’”
echo “Current Directory = $(pwd)”
echo "Network Name and Node Name = $(uname -n)"
echo "Kernal Name =$(uname -s)"
echo "Kernal Version=$(uname -v)"
echo "Kernal Release =$(uname -r)"
echo "Kernal OS =$(uname -o)"
echo “Proessor Type = $(uname -p)”
echo “Kernel Machine Information = $(uname –m)”
echo "All Information =$(uname -a)"
```

7 a FCFS

```
#include <stdio.h>

struct Process {
    int id, arrivalTime, burstTime, waitingTime, turnaroundTime;
};

void findWaitingTime(struct Process p[], int n) {
    p[0].waitingTime = 0;
    for (int i = 1; i < n; i++)
        p[i].waitingTime = p[i - 1].waitingTime + p[i - 1].burstTime;
}

void findTurnaroundTime(struct Process p[], int n) {
    for (int i = 0; i < n; i++)
        p[i].turnaroundTime = p[i].waitingTime + p[i].burstTime;
}

void findAverageTime(struct Process p[], int n) {
    findWaitingTime(p, n);
    findTurnaroundTime(p, n);

    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", p[i].id, p[i].burstTime, p[i].waitingTime, p[i].turnaroundTime);
    }
}

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

    struct Process p[n];
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &p[i].burstTime);
    }

    findAverageTime(p, n);
    return 0;
}
```

7b #include <stdio.h>

```
#include <stdio.h>

struct Process {
    int id, burstTime, waitingTime, turnaroundTime;
};
```

```
void findWaitingTime(struct Process p[], int n) {
    p[0].waitingTime = 0;
    for (int i = 1; i < n; i++)
        p[i].waitingTime = p[i - 1].waitingTime + p[i - 1].burstTime;
}

void findTurnaroundTime(struct Process p[], int n) {
    for (int i = 0; i < n; i++)
        p[i].turnaroundTime = p[i].waitingTime + p[i].burstTime;
}

void sortByBurstTime(struct Process p[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i].burstTime > p[j].burstTime) {
                struct Process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
}

void findAverageTime(struct Process p[], int n) {
    sortByBurstTime(p, n);
    findWaitingTime(p, n);
    findTurnaroundTime(p, n);

    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", p[i].id, p[i].burstTime, p[i].waitingTime, p[i].turnaroundTime);
    }
}

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

    struct Process p[n];
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &p[i].burstTime);
    }

    findAverageTime(p, n);
    return 0;
}
```

7C priority queue

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

```
struct Process {
    int id, burstTime, priority, waitingTime, turnaroundTime;
};

void findWaitingTime(struct Process p[], int n) {
    p[0].waitingTime = 0;
    for (int i = 1; i < n; i++)
        p[i].waitingTime = p[i - 1].waitingTime + p[i - 1].burstTime;
}

void findTurnaroundTime(struct Process p[], int n) {
    for (int i = 0; i < n; i++)
        p[i].turnaroundTime = p[i].waitingTime + p[i].burstTime;
}

void sortByPriority(struct Process p[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (p[i].priority > p[j].priority) {
                struct Process temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }
    }
}

void findAverageTime(struct Process p[], int n) {
    sortByPriority(p, n);
    findWaitingTime(p, n);
    findTurnaroundTime(p, n);

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

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

    struct Process p[n];
    for (int i = 0; i < n; i++) {
        p[i].id = i + 1;
        printf("Enter burst time and priority for process %d: ", i + 1);
        scanf("%d %d", &p[i].burstTime, &p[i].priority);
    }

    findAverageTime(p, n);
    return 0;
}
```

8 reader writers

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

sem_t wrt;        // Semaphore to allow writer or reader access
pthread_mutex_t mutex;
int read_count = 0; // Track the number of readers

void *writer(void *arg) {
    sem_wait(&wrt);  // Writer enters critical section
    printf("Writer is writing\n");
    sem_post(&wrt);  // Writer exits critical section
    return NULL;
}

void *reader(void *arg) {
    pthread_mutex_lock(&mutex);
    read_count++;
    if (read_count == 1)
        sem_wait(&wrt);  // First reader locks the writer out
    pthread_mutex_unlock(&mutex);

    printf("Reader is reading\n");

    pthread_mutex_lock(&mutex);
    read_count--;
    if (read_count == 0)
        sem_post(&wrt);  // Last reader unlocks the writer
    pthread_mutex_unlock(&mutex);

    return NULL;
}

int main() {
    pthread_t r1, r2, w1;
    sem_init(&wrt, 0, 1);
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&r1, NULL, reader, NULL);
    pthread_create(&r2, NULL, reader, NULL);
    pthread_create(&w1, NULL, writer, NULL);

    pthread_join(r1, NULL);
    pthread_join(r2, NULL);
    pthread_join(w1, NULL);

    sem_destroy(&wrt);
    pthread_mutex_destroy(&mutex);
    return 0;
}
```

9.Dinners Philosphers

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

#define N 5  // Number of philosophers

sem_t chopstick[N];  // Semaphores for chopsticks

void *philosopher(void *num) {
    int id = *(int *)num;

    sem_wait(&chopstick[id]);              // Pick left chopstick
    sem_wait(&chopstick[(id + 1) % N]);    // Pick right chopstick

    printf("Philosopher %d is eating\n", id + 1);

    sem_post(&chopstick[id]);              // Put down left chopstick
    sem_post(&chopstick[(id + 1) % N]);    // Put down right chopstick

    printf("Philosopher %d is thinking\n", id + 1);
    return NULL;
}

int main() {
    pthread_t philosophers[N];
    int ids[N];

    for (int i = 0; i < N; i++)
        sem_init(&chopstick[i], 0, 1);  // Initialize chopstick semaphores

    for (int i = 0; i < N; i++) {
        ids[i] = i;
        pthread_create(&philosophers[i], NULL, philosopher, &ids[i]);
    }

    for (int i = 0; i < N; i++)
        pthread_join(philosophers[i], NULL);

    for (int i = 0; i < N; i++)
        sem_destroy(&chopstick[i]);  // Destroy semaphores

    return 0;
}
```

10. first fit, best fit , worst fit

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

```
#define MAX 5  // Maximum number of memory blocks

void firstFit(int blockSize[], int m, int processSize[], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                printf("First Fit: Process %d -> Block %d\n", i + 1, j + 1);
                blockSize[j] -= processSize[i];
                break;
            }
        }
    }
}

void bestFit(int blockSize[], int m, int processSize[], int n) {
    for (int i = 0; i < n; i++) {
        int bestIdx = -1;
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])
                    bestIdx = j;
            }
        }
        if (bestIdx != -1) {
            printf("Best Fit: Process %d -> Block %d\n", i + 1, bestIdx + 1);
            blockSize[bestIdx] -= processSize[i];
        }
    }
}

void worstFit(int blockSize[], int m, int processSize[], int n) {
    for (int i = 0; i < n; i++) {
        int worstIdx = -1;
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])
                    worstIdx = j;
            }
        }
        if (worstIdx != -1) {
            printf("Worst Fit: Process %d -> Block %d\n", i + 1, worstIdx + 1);
            blockSize[worstIdx] -= processSize[i];
        }
    }
}

int main() {
    int blockSize[MAX] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);
    firstFit(blockSize, m, processSize, n);
    bestFit(blockSize, m, processSize, n);
    worstFit(blockSize, m, processSize, n);

    return 0;
}
```

## 11. Bankers Algorithm

```
#include <stdio.h>

#define MAX_PROCESSES 5
#define MAX_RESOURCES 3

int allocate[MAX_PROCESSES][MAX_RESOURCES], max[MAX_PROCESSES][MAX_RESOURCES],
need[MAX_PROCESSES][MAX_RESOURCES];
int available[MAX_RESOURCES];

void calculateNeed() {
    for (int i = 0; i < MAX_PROCESSES; i++)
        for (int j = 0; j < MAX_RESOURCES; j++)
            need[i][j] = max[i][j] - allocate[i][j];
}

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

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

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

                if (j == MAX_RESOURCES) {
                    for (int k = 0; k < MAX_RESOURCES; k++)
                        work[k] += allocate[p][k];
                    safeSeq[count++] = p;
                    finish[p] = 1;
                    found = 1;
                }
            }
        }
        if (!found) {
            printf("System is not in a safe state.\n");
            return 0;
        }
    }

    printf("System is in a safe state.\nSafe sequence is: ");
    for (int i = 0; i < MAX_PROCESSES; i++)
        printf("%d ", safeSeq[i]);
    printf("\n");
    return 1;
}

int main() {
    // Example data (can be modified)
    int n, m;

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

    printf("Enter allocation matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &allocate[i][j]);

    printf("Enter maximum matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &max[i][j]);

    printf("Enter available resources:\n");
    for (int j = 0; j < m; j++)
        scanf("%d", &available[j]);

    calculateNeed();
    isSafe();

    return 0;
}
```

12 producer consumer

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

#define BUFFER_SIZE 5

int buffer[BUFFER_SIZE];
int count = 0;

sem_t empty, full;  // Semaphores
```

```
pthread_mutex_t mutex;

void *producer(void *arg) {
    for (int i = 0; i < 10; i++) {
        sem_wait(&empty);            // Wait for empty space
        pthread_mutex_lock(&mutex);  // Lock the buffer

        buffer[count++] = i;
        printf("Produced: %d\n", i);

        pthread_mutex_unlock(&mutex);  // Unlock the buffer
        sem_post(&full);               // Signal that the buffer has a new item
    }
    return NULL;
}

void *consumer(void *arg) {
    for (int i = 0; i < 10; i++) {
        sem_wait(&full);             // Wait for a full buffer
        pthread_mutex_lock(&mutex);  // Lock the buffer

        int item = buffer[--count];
        printf("Consumed: %d\n", item);

        pthread_mutex_unlock(&mutex);  // Unlock the buffer
        sem_post(&empty);              // Signal that the buffer has an empty space
    }
    return NULL;
}

int main() {
    pthread_t prod, cons;
    sem_init(&empty, 0, BUFFER_SIZE);  // Initialize empty semaphore
    sem_init(&full, 0, 0);             // Initialize full semaphore
    pthread_mutex_init(&mutex, NULL);  // Initialize mutex

    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);

    pthread_join(prod, NULL);
    pthread_join(cons, NULL);

    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

13. Best Memory allocation

```
#include <stdio.h>

#define MAX 5  // Maximum number of memory blocks
```

```
void firstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];

    for (int i = 0; i < n; i++) allocation[i] = -1; // Initialize all allocations to -1

    for (int i = 0; i < n; i++) {  // Process each process
        for (int j = 0; j < m; j++) {  // Find first fitting block
            if (blockSize[j] >= processSize[i]) {
                allocation[i] = j;
                blockSize[j] -= processSize[i];  // Reduce available block size
                break;
            }
        }
    }

    printf("Process No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

int main() {
    int blockSize[MAX] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);

    firstFit(blockSize, m, processSize, n);

    return 0;
}
```