OS Lab

Operating Systems

All 33 record programs · Algorithm, program and output

CONTENTS

01 Add two numbers 2
02 Simple interest 3
03 Odd or even 4
04 Factorial (loop) 5
05 Reverse a number 6
06 Sum of digits 7
07 Prime check 8
08 Fibonacci series 9
09 Area of a circle 10
10 GCD (Euclid) 11
11 Sum of a 3-digit number's digits 12
12 Odd & even sums (1–50) 13
13 Multiplication table 14
14 Circumference of a circle 15
15 FCFS scheduling (C) 16
16 Priority scheduling (C) 18
17 Leap year 20
18 Menu with case 21
19 Swap without a third variable 23
20 Grade from marks 24
21 Power without built-ins 25
22 Factorial with a function 26
23 Full name from title and name 27
24 Star pyramid 28
25 Number pattern 29
26 Array input & display 30
27 Largest in an array 31
28 Pascal's triangle 32
29 Largest of three 33
30 Linux commands 34
31 Round Robin scheduling 37
32 Banker's algorithm (deadlock avoidance) 39
33 First Fit, Best Fit, Worst Fit 42

PROGRAM 01

Add two numbers

Aim: Write a shell script to add the two numbers

ALGORITHM

```
1. Start
2. Echo a prompt, then read a
3. Read b the same way
4. Sum=$((a + b)) and echo the result
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the first number:"
read a

echo "Enter the second number:"
read b

sum=$((a + b))

echo "Sum of $a and $b = $sum"
```

INPUT

```
12
8
```

OUTPUT

```
Enter the first number:
12
Enter the second number:
8
Sum of 12 and 8 = 20
```

PROGRAM 02

Simple interest

Aim: Write a shell script to print the simple interest of an amount along with total amount

ALGORITHM

```
1. Start
2. Read principal p, rate r, time t
3. Si=$(echo "scale=2; $p * $r * $t / 100" | bc)
4. Total=$(echo "scale=2; $p + $si" | bc) and print both
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the principal amount:"
read p

echo "Enter the rate of interest (% per year):"
read r

echo "Enter the time (in years):"
read t

si=$(echo "scale=2; $p * $r * $t / 100" | bc)
total=$(echo "scale=2; $p + $si" | bc)

echo "Simple Interest = $si"
echo "Total Amount    = $total"
```

INPUT

```
5000
8
3
```

OUTPUT

```
Enter the principal amount:
5000
Enter the rate of interest (% per year):
8
Enter the time (in years):
3
Simple Interest = 1200.00
Total Amount    = 6200.00
```

PROGRAM 03

Odd or even

Aim: write a shell script to input a number and check whether it is an odd or even number.

ALGORITHM

```
1. Start
2. Read n
3. If [ $((n % 2)) -eq 0 ] → even
4. Else → odd, close with fi
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter a number:"
read n

if (( n % 2 == 0 ))
then
    echo "$n is an Even number"
else
    echo "$n is an Odd number"
fi
```

INPUT

```
7
```

OUTPUT

```
Enter a number:
7
7 is an Odd number
```

PROGRAM 04

Factorial (loop)

Aim: write a shell script to input a number and print the factorial of that number

ALGORITHM

```
1. Start
2. Read n, set fact=1
3. While [ $n -gt 1 ]: fact=$((fact * n)), n=$((n - 1))
4. Echo the factorial
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter a number:"
read n

fact=1
i=1

while (( i <= n ))
do
    fact=$((fact * i))
    i=$((i + 1))
done

echo "Factorial of $n = $fact"
```

INPUT

```
5
```

OUTPUT

```
Enter a number:
5
Factorial of 5 = 120
```

PROGRAM 05

Reverse a number

Aim: write a shell script to input a number and print the reverse of that number

ALGORITHM

```
1. Start
2. Read n, rev=0
3. Loop while n > 0: digit=$((n % 10)), rev=$((rev * 10 + digit)), n=$((n / 10))
4. Print rev
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter a number:"
read n

num=$n
rev=0

while (( num > 0 ))
do
    digit=$((num % 10))
    rev=$((rev * 10 + digit))
    num=$((num / 10))
done

echo "Reverse of $n = $rev"
```

INPUT

```
12345
```

OUTPUT

```
Enter a number:
12345
Reverse of 12345 = 54321
```

PROGRAM 06

Sum of digits

Aim: Write a shell script to print the sum of the digit of the input term.

ALGORITHM

```
1. Start
2. Read n, sum=0
3. While n > 0: sum += n % 10; n /= 10
4. Print sum
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter a number:"
read n

num=$n
sum=0

while (( num > 0 ))
do
    sum=$((sum + num % 10))
    num=$((num / 10))
done

echo "Sum of digits of $n = $sum"
```

INPUT

```
4567
```

OUTPUT

```
Enter a number:
4567
Sum of digits of 4567 = 22
```

PROGRAM 07

Prime check

Aim: write a shell script to input a number and check whether it is a prime number or not

ALGORITHM

```
1. Start
2. Read n; n < 2 is not prime
3. For i from 2 to n/2 (or √n): if n % i == 0 → not prime
4. Use a flag to print the result
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter a number:"
read n

if (( n < 2 ))
then
    echo "$n is not a Prime number"
    exit 0
fi

i=2
prime=1

while (( i * i <= n ))
do
    if (( n % i == 0 ))
    then
        prime=0
        break
    fi
    i=$((i + 1))
done

if (( prime == 1 ))
then
    echo "$n is a Prime number"
else
    echo "$n is not a Prime number"
fi
```

INPUT

```
29
```

OUTPUT

```
Enter a number:
29
29 is a Prime number
```

PROGRAM 08

Fibonacci series

Aim: write a shell script to input a term and print the Fibonacci Series of that term

ALGORITHM

```
1. Start
2. Read n terms, a=0, b=1
3. Loop n times: print a; next=$((a + b)); a=$b; b=$next
4. Echo -n keeps terms on one line
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the number of terms:"
read n

a=0
b=1

echo "Fibonacci Series of $n terms:"

for (( i = 1; i <= n; i++ ))
do
    echo -n "$a "
    c=$((a + b))
    a=$b
    b=$c
done

echo
```

INPUT

```
10
```

OUTPUT

```
Enter the number of terms:
10
Fibonacci Series of 10 terms:
0 1 1 2 3 5 8 13 21 34
```

PROGRAM 09

Area of a circle

Aim: write a shell script to input Radius and print the area of the circle

ALGORITHM

```
1. Start
2. Read r
3. Area=$(echo "scale=2; 3.14 * $r * $r" | bc)
4. Echo the area
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the radius of the circle:"
read r

area=$(echo "scale=2; 3.14 * $r * $r" | bc)

echo "Area of the circle = $area"
```

INPUT

```
7
```

OUTPUT

```
Enter the radius of the circle:
7
Area of the circle = 153.86
```

PROGRAM 10

GCD (Euclid)

Aim: Write a shell script to print the GCD of the two input terms

ALGORITHM

```
1. Start
2. Read a and b
3. While b != 0: r=$((a % b)); a=$b; b=$r
4. A is the GCD
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the first number:"
read a

echo "Enter the second number:"
read b

x=$a
y=$b

while (( y != 0 ))
do
    r=$((x % y))
    x=$y
    y=$r
done

echo "GCD of $a and $b = $x"
```

INPUT

```
48
18
```

OUTPUT

```
Enter the first number:
48
Enter the second number:
18
GCD of 48 and 18 = 6
```

PROGRAM 11

Sum of a 3-digit number's digits

Aim: Write the shell script to print the sum of the digit of the 3 digit number

ALGORITHM

```
1. Start
2. Read n (e.g. 456)
3. D1=$((n / 100)), d2=$((n / 10 % 10)), d3=$((n % 10))
4. Print d1 + d2 + d3
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter a 3 digit number:"
read n

if (( n < 100 || n > 999 ))
then
    echo "Please enter a 3 digit number"
    exit 1
fi

d1=$((n / 100))
d2=$((n / 10 % 10))
d3=$((n % 10))

sum=$((d1 + d2 + d3))

echo "$d1 + $d2 + $d3 = $sum"
echo "Sum of digits of $n = $sum"
```

INPUT

```
456
```

OUTPUT

```
Enter a 3 digit number:
456
4 + 5 + 6 = 15
Sum of digits of 456 = 15
```

PROGRAM 12

Odd & even sums (1–50)

Aim: Write a shell script to print the sum of odd and even numbers from 1 to 50

ALGORITHM

```
1. Start
2. For ((i=1; i<=50; i++))
3. If i % 2 == 0, add to even, else add to odd
4. Print both sums
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

odd=0
even=0

for (( i = 1; i <= 50; i++ ))
do
    if (( i % 2 == 0 ))
    then
        even=$((even + i))
    else
        odd=$((odd + i))
    fi
done

echo "Sum of odd numbers from 1 to 50  = $odd"
echo "Sum of even numbers from 1 to 50 = $even"
```

OUTPUT

```
Sum of odd numbers from 1 to 50  = 625
Sum of even numbers from 1 to 50 = 650
```

PROGRAM 13

Multiplication table

Aim: Write a shell script to print the multiplication table of the input term

ALGORITHM

```
1. Start
2. Read n
3. For i in 1..10: echo "$n x $i = $((n * i))"
4. Done
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter a number:"
read n

for (( i = 1; i <= 10; i++ ))
do
    echo "$n x $i = $((n * i))"
done
```

INPUT

```
7
```

OUTPUT

```
Enter a number:
7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

PROGRAM 14

Circumference of a circle

Aim: Write a shell script to print the circumference of the circle.

ALGORITHM

```
1. Start
2. Read r
3. C=$(echo "scale=2; 2 * 3.14 * $r" | bc)
4. Print c
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the radius of the circle:"
read r

circumference=$(echo "scale=2; 2 * 3.14 * $r" | bc)

echo "Circumference of the circle = $circumference"
```

INPUT

```
7
```

OUTPUT

```
Enter the radius of the circle:
7
Circumference of the circle = 43.96
```

PROGRAM 15

FCFS scheduling (C)

Aim: Write A Program in c to implement the FCFS Algorithm

ALGORITHM

```
1. Start
2. Read n and each burst time
3. Wt[0] = 0; wt[i] = wt[i-1] + bt[i-1]
4. Tat[i] = wt[i] + bt[i]; print the table and the averages
5. Stop
```

PROGRAM (C)

```c
#include <stdio.h>

int main()
{
    int n, i;
    int bt[20], wt[20], tat[20];
    float total_wt = 0, total_tat = 0;

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

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

    // First process does not wait
    wt[0] = 0;

    // Each process waits for all the ones before it
    for (i = 1; i < n; i++)
        wt[i] = wt[i - 1] + bt[i - 1];

    for (i = 0; i < n; i++)
    {
        tat[i] = wt[i] + bt[i];
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++)
        printf("P%d\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);

    printf("\nAverage Waiting Time    = %.2f\n", total_wt / n);
    printf("Average Turnaround Time = %.2f\n", total_tat / n);

    return 0;
}
```

INPUT

```
3
5
3
8
```

OUTPUT

```
Enter number of processes: 3
Enter burst time for each process:
P1: 5
P2: 3
P3: 8

Process Burst Time      Waiting Time    Turnaround Time
P1      5               0               5
P2      3               5               8
P3      8               8               16

Average Waiting Time    = 4.33
Average Turnaround Time = 9.67
```

PROGRAM 16

Priority scheduling (C)

Aim: Write a program in c to implement the priority bases scheduling algorithm

ALGORITHM

```
1. Start
2. Read burst time and priority for each process
3. Sort by priority (smaller number = higher priority), swapping bt and pid too
4. Compute wt and tat as in FCFS and print the averages
5. Stop
```

PROGRAM (C)

```c
// A lower number means a higher priority

#include <stdio.h>

int main()
{
    int n, i, j, temp;
    int p[20], bt[20], pr[20], wt[20], tat[20];
    float total_wt = 0, total_tat = 0;

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

    printf("Enter burst time and priority for each process:\n");
    for (i = 0; i < n; i++)
    {
        printf("P%d: ", i + 1);
        scanf("%d %d", &bt[i], &pr[i]);
        p[i] = i + 1;
    }

    // Sort processes by priority (selection sort)
    for (i = 0; i < n - 1; i++)
    {
        for (j = i + 1; j < n; j++)
        {
            if (pr[j] < pr[i])
            {
                temp = pr[i]; pr[i] = pr[j]; pr[j] = temp;
                temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
                temp = p[i];  p[i] = p[j];   p[j] = temp;
            }
        }
    }

    wt[0] = 0;
    for (i = 1; i < n; i++)
        wt[i] = wt[i - 1] + bt[i - 1];

    for (i = 0; i < n; i++)
    {
        tat[i] = wt[i] + bt[i];
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("\nProcess\tPriority\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++)
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", p[i], pr[i], bt[i], wt[i], tat[i]);

    printf("\nAverage Waiting Time    = %.2f\n", total_wt / n);
```

```
    printf("Average Turnaround Time = %.2f\n", total_tat / n);

    return 0;
}
```

INPUT

```
4
10 3
1 1
2 4
5 2
```

OUTPUT

```
Enter number of processes: 4
Enter burst time and priority for each process:
P1: 10 3
P2: 1 1
P3: 2 4
P4: 5 2

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

Average Waiting Time    = 5.75
Average Turnaround Time = 10.25
```

PROGRAM 17

Leap year

Aim: Write the shell script to input a year and check whether it is leap year or not

ALGORITHM

```
1. Start
2. Read y
3. Leap if (y % 4 == 0 && y % 100 != 0) || y % 400 == 0
4. Echo the result
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter a year:"
read y

if (( (y % 4 == 0 && y % 100 != 0) || y % 400 == 0 ))
then
    echo "$y is a Leap Year"
else
    echo "$y is not a Leap Year"
fi
```

INPUT

```
2024
```

OUTPUT

```
Enter a year:
2024
2024 is a Leap Year
```

PROGRAM 18

Menu with case

Aim: Write the shell script using infinite loop to multiply and subtract the two number using case statement

ALGORITHM

```
1. Start
2. While true: show the menu, read choice
3. Case $choice in 1) multiply ;; 2) subtract ;; 3) exit 0 ;; *) invalid ;; esac
4. Done
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

while true
do
    echo
    echo "1. Multiply"
    echo "2. Subtract"
    echo "3. Exit"
    echo "Enter your choice:"
    read choice

    case $choice in
        1)
            echo "Enter two numbers:"
            read a b
            echo "$a x $b = $((a * b))"
            ;;
        2)
            echo "Enter two numbers:"
            read a b
            echo "$a - $b = $((a - b))"
            ;;
        3)
            echo "Exiting..."
            exit 0
            ;;
        *)
            echo "Invalid choice"
            ;;
    esac
done
```

INPUT

```
1
6 7
2
20 8
3
```

OUTPUT

```


1. Multiply
2. Subtract
3. Exit
Enter your choice:
1
Enter two numbers:
```

```
6 7
6 x 7 = 42

1. Multiply
2. Subtract
3. Exit
Enter your choice:
2
Enter two numbers:
20 8
20 - 8 = 12

1. Multiply
2. Subtract
3. Exit
Enter your choice:
3
Exiting...
```

PROGRAM 19

Swap without a third variable

Aim: Write a program in shell script to swap the values without using the third variables

ALGORITHM

```
1. Start
2. Read a and b
3. A=$((a + b)); b=$((a - b)); a=$((a - b))
4. Print the swapped values
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the first number (a):"
read a

echo "Enter the second number (b):"
read b

echo "Before swapping: a = $a, b = $b"

a=$((a + b))
b=$((a - b))
a=$((a - b))

echo "After swapping:  a = $a, b = $b"
```

INPUT

```
10
20
```

OUTPUT

```
Enter the first number (a):
10
Enter the second number (b):
20
Before swapping: a = 10, b = 20
After swapping:  a = 20, b = 10
```

PROGRAM 20

Grade from marks

Aim: Write the shell script to print the grade after input the marks of four different subjects

ALGORITHM

```
1. Start
2. Read m1 m2 m3 m4 on one line
3. Avg=$(( (m1 + m2 + m3 + m4) / 4 ))
4. If / elif ladder: ≥ 90 → A, ≥ 75 → B, …
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the marks of 4 subjects (out of 100):"
read m1 m2 m3 m4

total=$((m1 + m2 + m3 + m4))
percent=$((total / 4))

if (( percent >= 90 ))
then
    grade="A+"
elif (( percent >= 80 ))
then
    grade="A"
elif (( percent >= 70 ))
then
    grade="B"
elif (( percent >= 60 ))
then
    grade="C"
elif (( percent >= 50 ))
then
    grade="D"
else
    grade="F"
fi

echo "Total      = $total / 400"
echo "Percentage = $percent%"
echo "Grade      = $grade"
```

INPUT

```
85 90 78 92
```

OUTPUT

```
Enter the marks of 4 subjects (out of 100):
85 90 78 92
Total      = 345 / 400
Percentage = 86%
Grade      = A
```

PROGRAM 21

Power without built-ins

Aim: write the shell script to print the power of an input number and bash without using any predefined function

ALGORITHM

```
1. Start
2. Read base and exp
3. Result=1; loop exp times: result=$((result * base))
4. Print result
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the base:"
read base

echo "Enter the exponent:"
read exp

result=1

for (( i = 1; i <= exp; i++ ))
do
    result=$((result * base))
done

echo "$base ^ $exp = $result"
```

INPUT

```
2
10
```

OUTPUT

```
Enter the base:
2
Enter the exponent:
10
2 ^ 10 = 1024
```

PROGRAM 22

Factorial with a function

Aim: shell function to input a number and print the factorial of that number

ALGORITHM

```
1. Start
2. Factorial() { local n=$1; … echo $fact; }
3. Read num
4. Result=$(factorial $num) and print
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

factorial()
{
    local n=$1
    local fact=1

    for (( i = 2; i <= n; i++ ))
    do
        fact=$((fact * i))
    done

    echo $fact
}

echo "Enter a number:"
read n

echo "Factorial of $n = $(factorial $n)"
```

INPUT

```
6
```

OUTPUT

```
Enter a number:
6
Factorial of 6 = 720
```

PROGRAM 23

Full name from title and name

Aim: shell script to print full name after input name and title separately

ALGORITHM

```
1. Start
2. Read title (Mr / Ms / Dr)
3. Read name
4. Echo "$title. $name"
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the title (Mr/Ms/Dr):"
read title

echo "Enter the name:"
read name

full_name="$title. $name"

echo "Full Name: $full_name"
```

INPUT

```
Mr
John Smith
```

OUTPUT

```
Enter the title (Mr/Ms/Dr):
Mr
Enter the name:
John Smith
Full Name: Mr. John Smith
```

PROGRAM 24

Star pyramid

Aim: shell script to print pyramid

ALGORITHM

```
1. Start
2. Read rows n
3. For each row i: print n − i spaces, then 2i − 1 stars
4. Echo after each row
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the number of rows:"
read n

for (( i = 1; i <= n; i++ ))
do
    # Leading spaces
    for (( s = 1; s <= n - i; s++ ))
    do
        echo -n " "
    done

    # Stars
    for (( j = 1; j <= 2 * i - 1; j++ ))
    do
        echo -n "*"
    done

    echo
done
```

INPUT

```
5
```

OUTPUT

```
Enter the number of rows:
5
    *
   ***
  *****
 *******
*********
```

PROGRAM 25

Number pattern

Aim: Write the shell script to print the following pattern after input term #n=3

ALGORITHM

```
1. Start
2. Read n
3. Nested loops: outer i = 1..n, inner j = 1..i
4. Echo -n $j, then a newline after each row
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash
# For n = 3:
# 1
# 1 2
# 1 2 3

echo "Enter the number of terms:"
read n

for (( i = 1; i <= n; i++ ))
do
    for (( j = 1; j <= i; j++ ))
    do
        echo -n "$j "
    done
    echo
done
```

INPUT

```
3
```

OUTPUT

```
Enter the number of terms:
3
1
1 2
1 2 3
```

PROGRAM 26

Array input & display

Aim: Write a shell script to input the array elements and display it.

ALGORITHM

```
1. Start
2. Read n
3. For i in 0..n-1: read arr[i]
4. Loop again and print arr[$i] (or ${arr[@]})
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the number of elements:"
read n

echo "Enter $n elements:"
for (( i = 0; i < n; i++ ))
do
    read arr[i]
done

echo "The array elements are:"
for (( i = 0; i < n; i++ ))
do
    echo "arr[$i] = ${arr[i]}"
done
```

INPUT

```
5
10
20
30
40
50
```

OUTPUT

```
Enter the number of elements:
5
Enter 5 elements:
10
20
30
40
50
The array elements are:
arr[0] = 10
arr[1] = 20
arr[2] = 30
arr[3] = 40
arr[4] = 50
```

PROGRAM 27

Largest in an array

Aim: Write a shell script to print the largest element from an array

ALGORITHM

```
1. Start
2. Read the elements into arr
3. Max=${arr[0]}
4. Loop: if [ ${arr[i]} -gt $max ] then max=${arr[i]}
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the number of elements:"
read n

echo "Enter $n elements:"
for (( i = 0; i < n; i++ ))
do
    read arr[i]
done

max=${arr[0]}

for (( i = 1; i < n; i++ ))
do
    if (( arr[i] > max ))
    then
        max=${arr[i]}
    fi
done

echo "Array: ${arr[*]}"
echo "Largest element = $max"
```

INPUT

```
5
12
45
7
89
23
```

OUTPUT

```
Enter the number of elements:
5
Enter 5 elements:
12
45
7
89
23
Array: 12 45 7 89 23
Largest element = 89
```

PROGRAM 28

Pascal's triangle

Aim: Write a shell script to input a term and print the pascal triangle of that term

ALGORITHM

```
1. Start
2. Read rows
3. Each entry: c = c × (i − j) / (j + 1), starting at 1
4. Print spaces first to centre each row
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter the number of terms:"
read n

for (( i = 0; i < n; i++ ))
do
    # Leading spaces to center the row
    for (( s = 0; s < n - i - 1; s++ ))
    do
        echo -n "  "
    done

    # Each value is C(i, j) = C(i, j - 1) * (i - j + 1) / j
    c=1
    for (( j = 0; j <= i; j++ ))
    do
        printf "%4d" $c
        c=$((c * (i - j) / (j + 1)))
    done

    echo
done
```

INPUT

```
5
```

OUTPUT

```
Enter the number of terms:
5
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

PROGRAM 29

Largest of three

Aim: shell script to print the largest of three input numbers

ALGORITHM

```
1. Start
2. Read a b c
3. If a ≥ b and a ≥ c → a; elif b ≥ c → b; else c
4. Print the largest
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter three numbers:"
read a b c

if (( a >= b && a >= c ))
then
    largest=$a
elif (( b >= a && b >= c ))
then
    largest=$b
else
    largest=$c
fi

echo "Largest of $a, $b and $c = $largest"
```

INPUT

```
15 42 27
```

OUTPUT

```
Enter three numbers:
15 42 27
Largest of 15, 42 and 27 = 42
```

PROGRAM 30

Linux commands

Aim: Linux commands such as mkdir,grep,sed,cal,cat,cd,xclock,oclock,>>(append),ls etc

ALGORITHM

```
1. Start
2. Mkdir, cd, ls: make and move around folders
3. > writes, >> appends; cat shows a file
4. Grep searches, sed replaces, cal shows a calendar; xclock / oclock need X11
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash
# mkdir, cd, ls, cat, >, >> (append), grep, sed, cal, xclock, oclock
# Everything runs inside a throwaway folder, so nothing else is touched.

work=$(mktemp -d)

echo "== mkdir: create a directory =="
mkdir "$work/demo"
echo "Created demo/"

echo
echo "== cd: change directory =="
cd "$work/demo"
echo "Now in: $(basename "$PWD")"

echo
echo "== > (write) and >> (append) =="
echo "apple" > fruits.txt
echo "banana" >> fruits.txt
echo "cherry" >> fruits.txt
echo "mango" >> fruits.txt
echo "Wrote 1 line with > and appended 3 with >>"

echo
echo "== ls: list files =="
ls

echo
echo "== cat: show file contents =="
cat fruits.txt

echo
echo "== grep: search for lines containing 'an' =="
grep "an" fruits.txt

echo
echo "== sed: replace 'apple' with 'grape' =="
sed 's/apple/grape/' fruits.txt

echo
echo "== cal: calendar for July 2026 =="
cal 7 2026

echo
echo "== xclock / oclock: graphical clocks (need X11) =="
for clock in xclock oclock
do
    if command -v $clock > /dev/null 2>&1 && [ -n "$DISPLAY" ]
    then
        echo "Starting $clock (close its window to continue)"
        $clock
```

```
    else
        echo "$clock: not available here (install x11-apps and run inside X11)"
    fi
done

cd /
rm -rf "$work"
```

OUTPUT

```
== mkdir: create a directory ==
Created demo/

== cd: change directory ==
Now in: demo

== > (write) and >> (append) ==
Wrote 1 line with > and appended 3 with >>

== ls: list files ==
fruits.txt

== cat: show file contents ==
apple
banana
cherry
mango

== grep: search for lines containing 'an' ==
banana
mango

== sed: replace 'apple' with 'grape' ==
grape
banana
cherry
mango

== cal: calendar for July 2026 ==
     July 2026
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31


== xclock / oclock: graphical clocks (need X11) ==
Starting xclock (close its window to continue)

  [ an xclock window opens ]
  +--------- xclock ---------+
  |            12            |
  |      11    |     1       |
  |   10       |        2    |
  |            |             |
  |  9         o-------  3   |
  |                         |
  |   8                  4  |
  |      7           5      |
  |            6            |
  +--------------------------+

Starting oclock (close its window to continue)

  [ an oclock window opens ]
        .-"""""""""-.
```

```
       .'     |      '.
      /       |        \
     ;        |         ;
     |        o------   |
     ;                  ;
      \                /
       '.            .'
         '-........-'

(On a machine without X11 the script prints instead:)
xclock: not available here (install x11-apps and run inside X11)
oclock: not available here (install x11-apps and run inside X11)
```

PROGRAM 31

Round Robin scheduling

Aim: Write A Program in shell script to implement the round robin scheduling Algorithm

ALGORITHM

```
1. Start
2. Read n, each burst time into bt[i] (copy into rem[i]), and the time quantum q
3. Loop while processes are left: each one with rem[i] > 0 runs min(q, rem[i]); when rem[i] hits
 0, ct[i] = time
4. Tat = ct (all arrive at 0), wt = tat − bt; print the Gantt chart, the table and both averages
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter number of processes:"
read n

echo "Enter burst time of each process:"
for (( i = 0; i < n; i++ ))
do
    read bt[i]
    rem[i]=${bt[i]}
done

echo "Enter time quantum:"
read q

time=0
left=$n
gantt="0"

# Keep cycling through the ready queue until every process is done
while (( left > 0 ))
do
    for (( i = 0; i < n; i++ ))
    do
        if (( rem[i] > 0 ))
        then
            if (( rem[i] > q ))
            then
                time=$((time + q))
                rem[i]=$((rem[i] - q))
            else
                time=$((time + rem[i]))
                rem[i]=0
                ct[i]=$time
                left=$((left - 1))
            fi
            gantt="$gantt | P$((i + 1)) | $time"
        fi
    done
done

echo
echo "Gantt chart: $gantt"
echo

total_wt=0
total_tat=0
printf "%-10s%-8s%-8s%s\n" "Process" "BT" "WT" "TAT"
for (( i = 0; i < n; i++ ))
do
```

```
    tat[i]=${ct[i]}
    wt[i]=$((tat[i] - bt[i]))
    total_wt=$((total_wt + wt[i]))
    total_tat=$((total_tat + tat[i]))
    printf "%-10s%-8s%-8s%s\n" "P$((i + 1))" "${bt[i]}" "${wt[i]}" "${tat[i]}"
done

echo
echo "Average Waiting Time    = $(echo "scale=2; $total_wt / $n" | bc)"
echo "Average Turnaround Time = $(echo "scale=2; $total_tat / $n" | bc)"
```

INPUT

```
3
5
3
8
4
```

OUTPUT

```
Enter number of processes:
3
Enter burst time of each process:
5
3
8
Enter time quantum:
4

Gantt chart: 0 | P1 | 4 | P2 | 7 | P3 | 11 | P1 | 12 | P3 | 16

Process   BT      WT      TAT
P1        5       7       12
P2        3       4       7
P3        8       8       16

Average Waiting Time    = 6.33
Average Turnaround Time = 11.66
```

PROGRAM 32

Banker's algorithm (deadlock avoidance)

Aim: Write shell script for Bankers Algorithm.

ALGORITHM

```
1. Start
2. Read n, m, the Allocation and Max matrices (row by row) and Available
3. Need[i][j] = Max[i][j] − Allocation[i][j]; Work = Available, Finish[i] = 0
4. Repeat: find a process with Finish = 0 and Need ≤ Work; add its Allocation to Work, mark it f
inished, append it to the safe sequence. All finished → SAFE, else UNSAFE
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter number of processes:"
read n
echo "Enter number of resource types:"
read m

# 2D matrices are stored in 1D arrays: row i, column j is at index i * m + j
echo "Enter Allocation matrix (one row per process):"
for (( i = 0; i < n; i++ ))
do
    read line
    j=0
    for x in $line
    do
        alloc[i * m + j]=$x
        j=$((j + 1))
    done
done

echo "Enter Max matrix (one row per process):"
for (( i = 0; i < n; i++ ))
do
    read line
    j=0
    for x in $line
    do
        max[i * m + j]=$x
        j=$((j + 1))
    done
done

echo "Enter Available resources:"
read line
j=0
for x in $line
do
    work[j]=$x
    j=$((j + 1))
done

# Need = Max - Allocation
echo
echo "Need matrix:"
for (( i = 0; i < n; i++ ))
do
    row=""
    for (( j = 0; j < m; j++ ))
    do
        need[i * m + j]=$((max[i * m + j] - alloc[i * m + j]))
```

```
        row="$row ${need[i * m + j]}"
    done
    echo "P$i:$row"
    finish[i]=0
done

# Safety algorithm
seq=""
count=0
found=1
while (( count < n && found == 1 ))
do
    found=0
    for (( i = 0; i < n; i++ ))
    do
        if (( finish[i] == 0 ))
        then
            ok=1
            for (( j = 0; j < m; j++ ))
            do
                if (( need[i * m + j] > work[j] ))
                then
                    ok=0
                fi
            done

            if (( ok == 1 ))
            then
                # Process i can finish, so it gives back what it holds
                for (( j = 0; j < m; j++ ))
                do
                    work[j]=$((work[j] + alloc[i * m + j]))
                done
                finish[i]=1
                seq="$seq P$i"
                count=$((count + 1))
                found=1
            fi
        fi
    done
done

echo
if (( count == n ))
then
    echo "The system is in a SAFE state."
    echo "Safe sequence:$seq"
else
    echo "The system is in an UNSAFE state (deadlock possible)."
fi
```

INPUT

```
5
3
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
3 3 2
```

OUTPUT

```
Enter number of processes:
5
Enter number of resource types:
3
Enter Allocation matrix (one row per process):
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter Max matrix (one row per process):
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter Available resources:
3 3 2

Need matrix:
P0: 7 4 3
P1: 1 2 2
P2: 6 0 0
P3: 0 1 1
P4: 4 3 1

The system is in a SAFE state.
Safe sequence: P1 P3 P4 P0 P2
```

PROGRAM 33

First Fit, Best Fit, Worst Fit

Aim: Write shell script for the following memory management methods, First Fit, Best Fit, and Worst fit.

ALGORITHM

```
1. Start
2. Read the block sizes into block[] and the process sizes into proc[]
3. Allocate() copies block[] to hole[]; for each process scan every block that is big enough: Fi
rst Fit takes the first, Best Fit the smallest, Worst Fit the largest
4. Subtract the process size from that block (hole[pick] −= size), print the block and what is l
eft, or 'Not allocated'
5. Stop
```

PROGRAM (SHELL SCRIPT)

```
#!/bin/bash

echo "Enter number of memory blocks:"
read nb
echo "Enter size of each block:"
for (( j = 0; j < nb; j++ ))
do
    read block[j]
done

echo "Enter number of processes:"
read np
echo "Enter size of each process:"
for (( i = 0; i < np; i++ ))
do
    read proc[i]
done

# $1 is the method name: First Fit, Best Fit or Worst Fit
allocate()
{
    # Every method starts again with all the blocks free
    for (( j = 0; j < nb; j++ ))
    do
        hole[j]=${block[j]}
    done

    echo
    echo "---------- $1 ----------"
    printf "%-10s%-8s%-8s%s\n" "Process" "Size" "Block" "Left"

    for (( i = 0; i < np; i++ ))
    do
        pick=-1
        for (( j = 0; j < nb; j++ ))
        do
            if (( hole[j] >= proc[i] ))
            then
                if (( pick == -1 ))
                then
                    pick=$j
                    # First Fit stops at the first block that is big enough
                    if [ "$1" = "First Fit" ]
                    then
                        break
                    fi
                elif [ "$1" = "Best Fit" ] && (( hole[j] < hole[pick] ))
                then
```

```
                    pick=$j    # smallest block that fits
                elif [ "$1" = "Worst Fit" ] && (( hole[j] > hole[pick] ))
                then
                    pick=$j    # largest block that fits
                fi
            fi
        done

        if (( pick == -1 ))
        then
            printf "%-10s%-8s%s\n" "P$((i + 1))" "${proc[i]}" "Not allocated"
        else
            hole[pick]=$((hole[pick] - proc[i]))
            printf "%-10s%-8s%-8s%s\n" "P$((i + 1))" "${proc[i]}" "B$((pick + 1))" "${hole[pick]}"
        fi
    done
}

allocate "First Fit"
allocate "Best Fit"
allocate "Worst Fit"
```

INPUT

```
5
100
500
200
300
600
4
212
417
112
426
```

OUTPUT

```
Enter number of memory blocks:
5
Enter size of each block:
100
500
200
300
600
Enter number of processes:
4
Enter size of each process:
212
417
112
426

---------- First Fit ----------
Process   Size    Block   Left
P1        212     B2      288
P2        417     B5      183
P3        112     B2      176
P4        426     Not allocated

---------- Best Fit ----------
Process   Size    Block   Left
P1        212     B4      88
P2        417     B2      83
P3        112     B3      88
P4        426     B5      174
```

```
---------- Worst Fit ----------
Process   Size    Block   Left
P1        212     B5      388
P2        417     B2      83
P3        112     B5      276
P4        426     Not allocated
```