1. Greatest of Three Numbers:

```
bash
#!/bin/bash
echo "Enter three numbers:"
read num1
read num2
read num3
if [ "$num1" -gt "$num2" ] && [ "$num1" -gt "$num3" ]; then
   echo "The greatest number is: $num1"
elif [ "$num2" -gt "$num1" ] && [ "$num2" -gt "$num3" ]; then
   echo "The greatest number is: $num2"
else
   echo "The greatest number is: $num3"
fi
```

///
//////////////////////////////////

2. Factorial of N Numbers:

```
bash
#!/bin/bash
echo "Enter a number:"
read num
factorial=1
for ((i=1; i<=num; i++)); do
   factorial=$((factorial * i))
done
echo "Factorial of $num is: $factorial"
```

///
//////////////////////////////////

3. Sum of N Numbers:

```
bash
#!/bin/bash
echo "Enter the count of numbers:"
read count
sum=0
echo "Enter $count numbers:"
for ((i=1; i<=count; i++)); do
   read num
   sum=$((sum + num))
done
echo "Sum of the numbers is: $sum"
/////////////////////////////////////////////////////////////////
//////////////////////////////////
```

4. Number is Odd or Even:

```
bash
#!/bin/bash
echo "Enter a number:"
read num
if [ $((num % 2)) -eq 0 ]; then
   echo "$num is even."
else
   echo "$num is odd."
fi
/////////////////////////////////////////////////////////////////
//////////////////////////////////
```

5. Fibonacci Series:

bash

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

///
//////////////////////////////////

6. Multiplication Table:

bash

```
#!/bin/bash
echo "Enter a number for the multiplication table:"
read num
echo "Multiplication table for $num:"
for ((i=1; i<=10; i++)); do
   echo "$num x $i = $((num * i))"
done
```

///
//////////////////////////////////

7. Swapping of Two Numbers:

bash

```bash
#!/bin/bash
echo "Enter two numbers:"
read num1
read num2
echo "Before swapping: num1=$num1, num2=$num2"
temp=$num1
num1=$num2
num2=$temp
echo "After swapping: num1=$num1, num2=$num2"
```

///
//////////////////////////////////

8. Palindrome Check:

bash

```bash
#!/bin/bash
echo "Enter a string or number:"
read input
reverse=$(echo $input | rev)
if [ "$input" = "$reverse" ]; then
   echo "$input is a palindrome."
else
   echo "$input is not a palindrome."
fi
```

///
//////////////////////////////////

9. Positive or Negative Number:

```
bash
#!/bin/bash
echo "Enter a number:"
read num
if [ $num -gt 0 ]; then
   echo "$num is a positive number."
elif [ $num -lt 0 ]; then
   echo "$num is a negative number."
else
   echo "$num is zero."
fi
/////////////////////////////////////////////////////////////////
//////////////////////////////////
```

10. Area of Different Shapes:

bash

```
#!/bin/bash
echo "Choose a shape (1. Circle, 2. Rectangle, 3. Triangle):"
read choice
case $choice in
  1)
    echo "Enter the radius of the circle:"
    read radius
    area=$(echo "3.14159 * $radius * $radius" | bc)
    echo "Area of the circle: $area"
    ;;
  2)
    echo "Enter the length of the rectangle:"
    read length
    echo "Enter the width of the rectangle:"
    read width
    area=$((length * width))
    echo "Area of the rectangle: $area"
    ;;
  3)
    echo "Enter the base of the triangle:"
    read base
    echo "Enter the height of the triangle:"
    read height
    area=$(echo "0.5 * $base * $height" | bc)
    echo "Area of the triangle: $area"
    ;;
  *)
    echo "Invalid choice."
    ;;
esac
```

//
//////////////////////////////////

1. Implementing LS System Calls:

c

```
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>
int main() {
    struct dirent *de;
    DIR *dr = opendir(".");
      if (dr == NULL) {
        printf("Could not open current directory\n");
        return 1;
    }
    printf("Files in current directory:\n");
    while ((de = readdir(dr)) != NULL)
        printf("%s\n", de->d_name);
    closedir(dr);
    return 0;
}
```

//////////////////////////////////.

2. Implementing Fork() System Calls:

```
#include <stdio.h>
#include <unistd.h>
int main() {
   pid_t pid = fork();
   if (pid == 0) {
      printf("Child process\n");
   } else if (pid > 0) {
      printf("Parent process\n");
   } else {
      printf("Fork failed\n");
      return 1;
   }
   return 0;
}
```

//////////////////////////////////

3. Implementing Open() System Calls:

```
#include <stdio.h>
#include <fcntl.h>
int main() {
   int fd = open("example.txt", O_CREAT | O_WRONLY | O_TRUNC, 0644);
   if (fd == -1) {
      printf("Error opening file\n");
      return 1;
   }
   write(fd, "Hello, Open() System Call!", 26);
   close(fd);
   return 0;
}
```

//
//////////////////////////////////

4. Implementing Write() System Calls:

c

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
int main() {
   int fd = open("example.txt", O_WRONLY | O_APPEND);
   if (fd == -1) {
      printf("Error opening file\n");
      return 1;
   }
   write(fd, " Appending text using write() System Call.", 40);
   close(fd);
   return 0;
}
```

//
//////////////////////////////////

5. Implementing Read() System Calls:

c

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
int main() {
    int fd = open("example.txt", O_RDONLY);
    if (fd == -1) {
        printf("Error opening file\n");
        return 1;
    }
    char buffer[100];
    read(fd, buffer, sizeof(buffer));
    close(fd);
    printf("Read from file: %s\n", buffer);
    return 0;
}
```

2. FCFS Disk Scheduling Algorithm

c

```
#include <stdio.h>
#include <stdlib.h>
void fcfsDiskScheduling(int requests[], int n, int initialPosition) {
   int totalSeekTime = 0;
   printf("Sequence of disk accesses: ");
   printf("%d ", initialPosition);
   for (int i = 0; i < n; i++) {
      totalSeekTime += abs(initialPosition - requests[i]);
      initialPosition = requests[i];
      printf("%d ", initialPosition);
   }
   printf("\nTotal Seek Time: %d\n", totalSeekTime);
}
int main() {
   int requests[] = {98, 183, 37, 122, 14, 124, 65, 67};
   int n = sizeof(requests) / sizeof(requests[0]);
   int initialPosition;
   printf("Enter the initial head position: ");
   scanf("%d", &initialPosition);
   fcfsDiskScheduling(requests, n, initialPosition);
   return 0;
}
```