**WEEK – 1**

**Problem Statement – 1**

Raju loves playing with maths. He is very fond of factorials. Now he is interested in knowing the nth number factorial. As he is not very good at programming, so he needs your help. Your task is to print the factorial of that number.

* **Constraints**

1. The input n is a positive number.
2. The input n <=16.

* **Explaination :**

Base case **:**

When n value becomes 0 or 1 , n value is returned

Recursive Case **:**

In recursive case we call the function to calculate the factorial of each preceding number until n value reaches zero

Ex : fact(5)

5\*fact(4)

5\*4\*fact(3)

5\*4\*3\*fact(2)

5\*4\*3\*2\*fact(1)

5\*4\*3\*2\*1

**Program :**

#include<stdio.h>

int fact(int x){

if(x<=1){

return x;

}

else{

return x\*fact(x-1); **// recursive call**

}

}

void main()

{

int n,res;

printf("Enter number to find the factorial : ");

scanf("%d",&n);

res=(fact(n));

printf("%d",res);

}

* **Sample Input & Output**

Enter a positive integer:7

Factorial of 7 = 5040

* **Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input** | **Excepted Output** | **Output after Execution** | **Test case Result : Pass/Fail** |
| TESTCASE-1 | 0 | 0 | 0 | Pass |
| TESTCASE-2 | 6 | 720 | 720 | Pass |
| TESTCASE-3 | 10 | 3628800 | 3628800 | Pass |

**Problem Statement:2**

Ms. Rekha wanted to have a party with her friends for her birthday. She brings m candies and n chocolates. She invited how many people to her birthday. Your job is to guide her towards the gcd, which contains m candies and n chocolates.

* **Constraints**

1. Input contains two integers m and n either positive or negative.
2. As any negative number is less than any positive number, the greatest common divisor must be positive
3. if m and n both are negative gcd is negative

* **Explaination :**

Base case :

When remainder becomes zero the recursion is terminated .

The Denominator (2nd value) in the the modulo operation is returned

Recursive case :

The Denominator(2nd value) and the remainder value are passed in recursive call .

Gcd(15,10) rem => 15%10 => 5

Since remainder is zero , 1 is returned .

GCD of 15,10 is 5

Gcd(15,2) rem => 15%2 => 1

Gcd(2,1) rem => 2%1 => 0

Since remainder is zero , 1 is returned .

GCD of 15,2 is 1

**Program :**

#include<stdio.h>

int gcd(int a,int b){

int rem=a%b;

if(rem==0){

return b;

}

else{

return gcd(b,rem); **// recursive call**

}

}

void main(){

int m,n,res;

printf("Enter m to find the GCD : ");

scanf("%d",&m);

printf("Enter n to find the GCD : ");

scanf("%d",&n);

res=(gcd(m,n));

printf("%d",res);

}

* **Sample Input & Output**

Enter the first number:128

Enter the second number:96

G.C.D of 128 and 96 is 32.

* **Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input** | **Excepted Output** | **Output after Execution** | **Test case Result : Pass/Fail** |
| TESTCASE-1 | m=366  n=60 | 6 | 6 | Pass |
| TESTCASE-2 | m=-20  n=100 | -20 | -20 | Pass |
| TESTCASE-3 | m=-2  n=1 | 1 | 1 | Pass |

**Problem Statement:3**

The first person holds '0' value and second person holds '1' value and third person adds the two values of the first person and second person. Now three persons will shift one position towards right hand side this process will continues until reaches maximum value of n. Now your task is to print above series(Fibonacci Series)

* **Constraints**

1. The input n>=1
2. The Fibonacci series nth value is 93

* **Explaination :**

The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1.

Base case :

When the value of n is less than 2 then n is returned

Recursive case :

Fibonacci(n-1) + Fibonacci(n-2) is returned

=> Fibonacci(4)

=> Fibonacci(3) + Fibonacci(2)

=>(Fibonacci(2) + Fibonacci(1)) + (Fibonacci(1) + Fibonacci(0))

=> ((Fibonacci(1) + Fibonacci(0)) + 1) + (1 + 0)

=> ((1 + 0) + 1) + (1 + 0)

=> 3 + 1

=> 4

**Program :**

#include<stdio.h>

int fibo(int n){

if(n<=1){

return n;

}

else{

return fibo(n-1)+fibo(n-2); **// recursive call**

}

}

void main()

{

int i,n,res;

printf("Enter number to find fibonacci sequence : ");

scanf("%d",&n);

if(n<=0){

printf("0\n1");

}

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

printf("%d\n",fibo(i));

}

}

* **Sample Input & Output**

Enter the number of elements:9

Fibonacci Series: 0 1 1 2 3 5 8 13 21

* **Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input** | **Excepted Output** | **Output after Execution** | **Test case Result : Pass/Fail** |
| TESTCASE-1 | 11 | 0 1 1 2 3 5 8 13 21 34 55 | 0 1 1 2 3 5 8 13 21 34 55 | Pass |
| TESTCASE-2 | -8 | 0 1 | 0 1 | Pass |
| TESTCASE-3 | 0 | 0 1 | 0 1 | Pass |

**Problem Statement:4**

Arjun wants to take the input from the keyboard and find the HCF(Highest Common Factor) of two given numbers. Your task is to find the logic of HCF of two numbers.

* **Constraints**

The input m and n are positive numbers

* **Explaination :**

Base case :

When remainder becomes zero the recursion is terminated .

The Denominator (2nd value) in the the modulo operation is returned

Recursive case :

The Denominator(2nd value) and the remainder value are passed in recursive call .

HCF(15,10) rem => 15%10 => 5

Since remainder is zero , 1 is returned .

HCF of 15,10 is 5

HCF(15,2) rem => 15%2 => 1

HCF(2,1) rem => 2%1 => 0

Since remainder is zero , 1 is returned .

HCF of 15,2 is 1

**Program :**

#include<stdio.h>

int hcf(int m,int n)

{

int rem=m%n;

if(rem==0)

return n;

else

return hcf(n,rem); **// recursive call**

}

void main()

{

int a,b,res;

printf("Enter a,b to find the HCF : ");

scanf("%d %d",&a,&b);

if(a>0 && b>0){

res=(hcf(a,b));

printf("%d",res);

}

else{

printf("Input must be positive number");

}

}

* **Sample Input & Output**

Enter any two numbers to find HCF:

12

30

HCF of 40 and 10 = 6

* **Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input** | **Excepted Output** | **Output after Execution** | **Test case Result : Pass/Fail** |
| TESTCASE-1 | a=40  b=10 | 10 | 10 | Pass |
| TESTCASE-2 | a=-36  b=60 | Input must be positive number | Input must be positive number | Pass |
| TESTCASE-3 | a=24  b=12 | 12 | 12 | Pass |

**Problem Statement:5**

In a college, Ms. Geetha teaches computer programming. She once assigned the students to print the first 50 natural numbers. Your job is to assist the students in discovering the rationale.

* **Constraints**
* Input n is a positive number and greater than 0.
* Input n=50
* **Explaination :**

Base case :

When count exceeds the n value program is terminated .

Recursive case :

In each each recursive call the count value is incremented until the count value reaches the n value

* If n is 50, the code uses the recursive function fun to print natural numbers up to n. The function takes two parameters: n and count.
* n is the maximum number we want to print up to.
* count represents the current number to be printed.
* If count is less than or equal to n (in this case 50), the number represented by count is printed.
* The function is then recursively called with count incremented by 1, effectively moving to the next natural number.
* This process continues until count exceeds n, at which point the recursive calls will stop due to the base condition (if(count <= n)).

**Program :**

#include<stdio.h>

void fun(int n , int count){

if(count > n){

return ;

}

else{

printf("%d ",count);

count=count+1;

fun(n,count); **// recursive call**

}

void main(){

int n,count=1;

printf("Enter the value of n :");

scanf("%d",&n);

if(n<=0){

printf("The ‘n’ value is not a negative number and must be 50");

}

else if(n!=50){

printf("The ‘n’ value must be 50");

}

else{

printf("The natural numbers are : ");

fun(n,count);

}

}

* **Sample Input & Output**

Enter n : 50

The natural numbers are : 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 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

* **Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case** | **Input** | **Excepted Output** | **Output after Execution** | **P/F** |
| CASE-1 | n=50 | The natural numbers are : 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 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 | The natural numbers are : 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 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 | Pass |
| CASE-2 | n=13 | The ‘n’ value must be 50 | The n value must be 50 | Pass |

**Problem Statement:6**

Mr. Satya came across a CodeChef website one day. He discovered a method to use recursion in the C programming language to count the digits of a given integer on that page. You need to assist Satya in understanding the logic.

* **Constraints**
* The n>0
* The n must be a less than or equal to any five digit number.
* **Explaination :**

Base case :

When n value becomes less than 0 , the count value is returned and program gets terminated .

Recursive case :

In each each recursive call , n is divided by 10 and count is returned .

* This recursive function is designed to count the number of digits in n. It takes two parameters: n and count.
* Inside the function, n is divided by 10 (n = n / 10). This effectively removes the last digit from n.
* If n is still greater than 0, it means there are more digits to count. So, count is incremented by 1, and the function is called recursively with the updated n and count.
* This process continues until n becomes 0, at which point the function returns the final count of digits.

**Program :**

#include<stdio.h>

int fun(int n , int count){

n=n/10;

if(n <= 0){

return count ;

}

else{

count=count+1 ;

return fun(n,count); **//recursive call**

}

}

void main(){

int n,count=1;

printf("Enter the value of n :");

scanf("%d",&n);

if(n==0 || n > 99999){

printf("The input should be less than or equal to five digit number. ");

}

else{

int res = fun(n,count);

printf("The number of digits in the number is : %d",res);

}

}

* **Sample Input & Output**

Input a number : 85

The number of digits in the number is : 2

* **Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input** | **Excepted Output** | **Output after Execution** | **Pass/Fail** |
| TESTCASE-1 | n=7 | The number of digits in the number is : 1 | The number of digits in the number is : 1 | Pass |
| TESTCASE-2 | n=53 | The number of digits in the number is : 2 | The number of digits in the number is : 2 | Pass |
| TESTCASE-3 | n=786 | The number of digits in the number is : 3 | The number of digits in the number is : 3 | Pass |
| TESTCASE-4 | n=144032 | The input should be less than or equal to five digit number. | The input should be less than or equal to five digit number. | Pass |

**Problem Statement:7**

The "Number Conversion Systems" Concept in Digital Logic Design is part of the first-year curriculum. The students wish to use their programming knowledge in C as well. Therefore, your objective is to assist in determining the reasoning behind utilizing recursion to convert decimal numbers to binary numbers.

* **Constraints**
* The n is a positive number
* The n<=three digit number

**Explanation :**

**1)Base Case:-**

The base case of the recursion is when **n** equals 0. In this case, the function simply returns without doing anything. This is because in binary, the representation of 0 is also 0, so there's nothing to print.

2)Recursive Case:-

**Recursive Case**: In the recursive case (when **n** is not 0), the following steps are taken:

* The function calculates the remainder of **n** divided by 2 using the modulo operator **%**. This remainder is the least significant binary digit (either 0 or 1).
* It prints this binary digit using **printf**.
* The function then calls itself with the argument **n/2**. This effectively discards the least significant digit and continues the process with the remaining part of the binary representation.
* This recursive process continues until **n** becomes 0, at which point the function returns without further recursion.

**Program:-**

#include<stdio.h>

void con(int n){

if(n==0)

return;

else{

printf("%d",n%2);

con(n/2);

}

}

void main(){

int n,a;

int count = 0;

printf("enter the number");

scanf("%d",&n);

a=n;

do {

a /= 10;

++count;

} while (a != 0);

if(n==0) printf("The binary number is 000");

else if(count>3) printf("Enter only upto three digit number!");

else{printf("The binary number is:");

con(n);

}

* **Sample Input & Output**

Enter a decimal number

9

The binary number is:1001

**Test Cases:-**

* **Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input** | **Excepted Output** | **Output after Execution** | **Test case Result : Pass/Fail** |
| TESTCASE-1 | n=7 | 111 | 111 | Pass |
| TESTCASE-2 | n=90 | 1011010 | 1011010 | Pass |
| TESTCASE-3 | n=553 | 1000101001 | 1000101001 | Pass |
| TESTCASE-4 | n=8933 | The n value must not exceed 3 digit number. | Enter only upto three digit number! | Pass |
| TESTCASE-5 | n=0 | 000 | 000 | Pass |

**Problem Statement:8**

                                 Ms. Parvathi was good at Mathematics. One day she wants to develop the program on LCM(Lease Common Multiple) of two numbers using C language. Your task is to help her to find the logic.

* **Constraints**
* The m,n are two positive numbers
* m,n>0
* The maximum input values is a 3 digit number.
* **Explanation:-**

Here when we call the lcm function it call the GCD function where the gcd of the two numbers is calculated. And to calculate the lcm we have a formula of (num1\*num2)/GCD(num1,num2)

In this way the LCM is calculated.

* **Program:-**

#include <stdio.h>

// Function to calculate GCD

int gcd(int a, int b) {

if (b == 0) {

return a;

} else {

return gcd(b, a % b);

}

}

// Function to calculate the LCM

int lcm(int m, int n) {

return (m \* n) / gcd(m, n);

}

int count(n){

int c=0;

while(n!=0)

{

n=n/10;

c++;

}

return c;

}

int main() {

int num1 ,num2;

printf("Enter two numbers: ");

scanf("%d%d",&num1,&num2);

if((num1==0 || num2==0)||(num1<0 || num2<0)) printf("Please Enter only positive numbers!");

else if(count(num1)>3|| count(num2)>3) printf("Enter only upto three digit number!");

else{

int result = lcm(num1, num2);

printf("LCM of %d and %d is %d\n", num1, num2, result);}

return 0;

}

* **Sample Input & Output**

Enter two numbers:

12

15

The LCM of 12 and 15 is 60

* **Testcases:-**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input** | **Excepted Output** | **Output after Execution** | **Test case Result : Pass/Fail** |
| TESTCASE-1 | m=2  n=3 | LCM=6 | LCM=6 | Pass |
| TESTCASE-2 | m=56  n=8 | LCM=8 | LCM=8 | Pass |
| TESTCASE-3 | m=-89  n=3 | Please enter positive value | Please enter positive value |  |
| TESTCASE-4 | n=36  n=-6 | Please enter positive value. | Please enter positive value | Pass |
| TESTCASE-5 | m=0  n=8 | The input value is not zero. | The input value is not zero. | Pass |
| TESTCASE-6 | m=4  n=0 | The input value is not zero. | The input value is not zero. | Pass |
| TESTCASE-7 | m=124  n=36 | LCM=1116 | LCM=1116 | Pass |
| TESTCASE-8 | m=7893  n=45 | The input is less than or equal to a 3 digit number | The input is less than or equal to a 3 digit number | Pass |

**Problem Statement: 9**

Mr. Ram was developing the idea of a function. He selected numbers from the keyboard from 1 to n. It is your responsibility to assist Ram in averaging one to n numbers and returning the result.

**Constraints:**

* ‘n’ is a positive integer (1<=n<=5)
* The result must be positive value.
* The average value is a positive value.

**Explanation:-**

1.**Base Case**:

* + The base case of the recursion is when i is equal to or greater than n. In this case, the function returns 0.
  + The base case essentially stops the recursive calls and starts unwinding the recursion.

1. **Recursive Case**:
   * In the recursive case (when i is less than n), the following steps are taken:
     + It adds the value of a[i] to the result of a recursive call to the count function with an incremented value of i. This means it adds the current element a[i] to the sum of the remaining elements in the array.
     + The recursion continues until i becomes equal to or greater than n.

**Program:-**

#include<stdio.h>

int count(int \*a,int n,int i){

if(i<n)

return a[i]+count(a,n,i+1);

else{

return 0;

}

}

void main(){

int a[5],n;

printf("Enter no. of elements:");

scanf("%d",&n);

if(n>5) printf("Enter utmost only 5 elements");

printf("\n Enter elements:");

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

scanf("%d", &a[i]);

}

float k = count(a,n,0);

if(k<0) printf("Enter only numbers so that result is positive!");

else {

float j= k/n;

printf ("\nAverage is %f ",j);}

}

**Output:-**

Enter no. of elements: 3

Enter elements**:** 1 2 3

Average is:2.0

**Test Cases:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case** | **Input1** | **Expected Output** | **Test case Result: Pass/Fail** | **Justification** |
| TC-1 | 1 | 1 | Pass | The sum is done and average is calculated |
| TC-2 | 7 3 1 | 3.66 | Pass | The sum is done and average is calculated |
| TC-3 | 12 14 17  18 20 50 | Out of range | Pass | Only 5 elements are allowed so out of range |
| TC-4 | 22 -9 25  7 | 11.25 | Pass | The sum is done and average is calculated |
| TC-4 | 0 | 0 | Pass | The sum is done and average is calculated |

**Problem Statement:10**

Mr. Kumar was a mathematical professor . He introduced a new concept, which was a triangular number. Now your task is to help Kumar , to check whether a given number is a triangular number or not. (Hint: Triangular numbers are those numbers which are obtained by continued summation of the natural numbers 1, 2, 3, 4, 5,etc.)(Note: Using Direct recursion)

**Constraints:**

* ‘n’ is a positive integer (1<=n<=25)

**Explanation:-**

1. **Base Case**:
   * If n is equal to 0, it returns 1. This is a special case where 0 is considered a triangular number (a triangle with 0 dots).
   * If n is negative, it returns 0. Negative numbers are not considered triangular numbers.
2. **Recursive Case**:
   * In all other cases (when n is a positive integer greater than 0), the function enters the recursive case.
   * It subtracts the current integer i from n and increments i by 1 in each recursive call.
   * The recursive call is made with the new values of n - i and i + 1.

**Program:-**

#include<stdio.h>

int num(n,i){

if (n == 0) {

return 1;

} else if (n < 0) {

return 0;

} else {

return num(n-i, i + 1);

}

}

void main(){

int n;

printf("Enter n:");

scanf("%d",&n);

if(n<0 || n>25 ) printf("Out of Bound");

else if(num(n,1)) printf("Trianglar number");

else {printf("Not a Triangular number");}

}

**Output:-**

Enter n:6

Triangular number

**Test Cases:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case** | **Input** | **Excepted**  **Output** | **Test case Result: Pass/Fail** | **Justification** |
| TC-1 | 28 | Out of bound | Pass | The function is called and number determined |
| TC-2 | 15 | Triangular Number | Pass | The function is called and number determined |
| TC-3 | 0 | is not a Triangular Number | Pass | The function is called and number determined |
| TC-4 | -10 | is not a Triangular Number | Pass | The function is called and number determined |

**Problem Statement:11**

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod.(Note: Towers of Hanoi implementation using recursive functions).

**Constraints:**

* Only one disk can be moved at a time.
* Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
* No disk may be placed on top of a smaller disk.
* ‘n’ is a number of disks in the Hanoi problem . It must be a positive number 1<=n<=5.

**Explanation:-**

1. **Base Case:**
   * The base case for the recursion is when there is only one disk to move (n == 1).
   * In the base case, it directly prints the step to move the disk from rod A to rod C using printf. This represents moving the single disk from the source rod to the destination rod.
2. **Recursive Case:**
   * In all other cases (when n is greater than 1), the function enters the recursive case.
   * It follows the classic steps of solving the Towers of Hanoi problem:
     1. Move n-1 disks from the source rod (A) to the auxiliary rod (B) using TOH(n-1, A, C, B). This step uses rod C as the auxiliary rod.
     2. Move the remaining one disk from the source rod (A) to the destination rod (C) using printf.
     3. Finally, move the n-1 disks from the auxiliary peg (B) to the destination rod (C) using TOH(n-1, B, A, C). This step uses rod A as the auxiliary p

**Program:-**

#include<stdio.h>

void TOH(int n,char A,char B,char C){

if (n==1) {

printf("\nmove disk from rod %c to rod %c",A,C);

return;

}

TOH(n-1,A,C,B);

printf("\nmove disk from rod %c to rod %c",A,C);

TOH(n-1,B,A,C);

}

void main(){

int n;

printf("Enter n:");

scanf("%d",&n);

TOH(n,'A','B','C');

}

**Output:-**

Enter n: 3

Move disk 1 from rod A to rod C

Move disk 2 from rod A to rod B

Move disk 1 from rod C to rod B

Move disk 3 from rod A to rod C

Move disk 1 from rod B to rod A

Move disk 2 from rod B to rod C

Move disk 1 from rod A to rod C

**Test Cases:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test case** | **Input1** | **Expected**  **Output** | **Test case Result: Pass/Fail** | **Justification** |
| TC-1 | 1 | Move disk 1 from rod A to rod C | Pass | Function is called and steps are displayed |
| TC-2 | -7 | Out of Size | Pass | Function is called and steps are displayed |
| TC-3 | 6 | Out of Size | Pass | Function is called and steps are displayed |
| TC-4 | 4 | Move disk 1 from rod A to rod B  Move disk 2 from rod A to rod C  Move disk 1 from rod B to rod C  Move disk 3 from rod A to rod B  Move disk 1 from rod C to rod A  Move disk 2 from rod C to rod B  Move disk 1 from rod A to rod B  Move disk 4 from rod A to rod C  Move disk 1 from rod B to rod C  Move disk 2 from rod B to rod A  Move disk 1 from rod C to rod A  Move disk 3 from rod B to rod C  Move disk 1 from rod A to rod B  Move disk 2 from rod A to rod C  Move disk 1 from rod B to rod C | Pass | Function is called and steps are displayed |