1. Write a function to find the factorial of a number.

IPO

| **Part** | **Description** |
| --- | --- |
| **Input** | A number n entered by the user |
| **Process** | Multiply numbers from 1 to n to get factorial |
| **Output** | Print the factorial of the number |

CODE

#include<stdio.h>

void fact()

{

int n,i,fact=1;

scanf("%d",&n);

for(i=1;i<=n;i++)

{

fact=fact\*i;

}

printf("%d",fact);

}

void main()

{

fact();

}

OUTPUT



1. Write a function to check whether a number is prime.

IPO

| **Part** | **Description** |
| --- | --- |
| **Input** | A number n from the user |
| **Process** | Count how many numbers divide n exactly |
| **Output** | Print if the number is prime or not |

CODE

#include<stdio.h>

void prime()

{

int n,i,count=0;

scanf("%d",&n);

for(i=1;i<=n;i++)

{

if(n%i==0)

count++;

}

if(count==2)

{

printf("its a prime number");

}

else

{

printf("its not a prime number");

}

}

void main()

{

prime();

}

OUTPUT



1. Write a function to calculate power using recursion

IPO

| **Input** | **Two integers: base b and exponent e** |
| --- | --- |
| **Process** | Multiply base b, e times using recursion |
| **Output** | Value of b^e (b to the power e) |

CODE

#include <stdio.h>

int power(int b, int e)

{

if(e == 0)

return 1;

return b\*power(b,e-1);

}

void main()

{

int b,e;

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

printf("%d",power(b,e));

}

OUTPUT



1. Write a function to check palindrome number using recursion.

IPO

| **Component** | **Description** |
| --- | --- |
| **I - Input** | - An integer a entered by the user |
| **P - Process** | - Calculate the reverse of the number using recursion - Compare reversed number with original - Decide if it is a palindrome |
| **O - Output** | - Print “its a palindrome” if equal - Else print “its not a palindrome” |

CODE

#include<stdio.h>

int reversenumber(int b,int reverse,int c)

{

int digit;

if(b==0)

return reverse;

digit=b%10;

reverse=reverse+digit\*c;

b=b/10;

c=c/10;

return reversenumber(b, reverse, c);;

}

void palindrome(int a)

{

int reverse = reversenumber(a, 0, 100);

if(reverse == a)

{

printf("its a palindrome");

}

else

{

printf("its not a palindrome");

}

}

void main()

{

int digit,reverse=0,a,b,c=100,i;

scanf("%d",&a);

palindrome(a);

}

OUTPUT



1. Write a function to calculate nCr (combinations).

IPO

| **Input** | **Process** | **Output** |
| --- | --- | --- |
| Two integers n and r | 1. Compute factorial of n → n!  2. Compute factorial of r → r!  3. Compute factorial of n - r → (n - r)!  4. Apply formula: nCr = n! / (r! \* (n - r)!) | Value of nCr (number of combinations) |

CODE

#include<stdio.h>

int factorial(int n)

{

int i, fact = 1;

for(i = 1; i <= n; i++)

{

fact = fact \* i;

}

return fact;

}

int nCr(int n, int r)

{

return factorial(n) / (factorial(r) \* factorial(n - r));

}

void main()

{

int n, r;

printf("Enter n and r: ");

scanf("%d%d", &n, &r);

if(r > n)

{

printf("Invalid input. r should be less than or equal to n.\n");

}

else

{

printf("nCr = %d\n", nCr(n, r));

}

}

OUTPUT



1. Write a program to demonstrate call by value and call by reference.

IPO

| **Component** | **Description** |
| --- | --- |
| **Input** | - Two integers (initial values, e.g., x = 5, y = 5) |
| **Process** | - callByValue: adds 10 to a copy of variable (does **not** affect original)  - callByReference: adds 10 directly to original using pointer (affects original) |
| **Output** | - Prints value before, inside, and after both function calls to show the effect |

CODE

#include <stdio.h>

// Call by Value

void callByValue(int a) {

a = a + 10;

printf("Inside callByValue: a = %d\n", a);

}

// Call by Reference

void callByReference(int \*b) {

\*b = \*b + 10;

printf("Inside callByReference: b = %d\n", \*b);

}

void main() {

int x = 5, y = 5;

printf("Before callByValue: x = %d\n", x);

callByValue(x);

printf("After callByValue: x = %d\n\n", x);

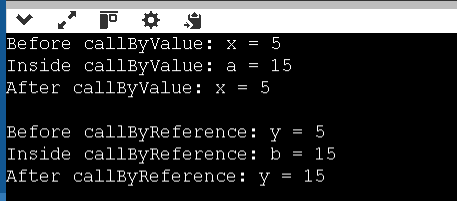
printf("Before callByReference: y = %d\n", y);

callByReference(&y);

printf("After callByReference: y = %d\n", y);

}

OUTPUT



1. Write a program using function to swap two numbers.

Ipo

| **Input** | **Two integers (x and y)** |
| --- | --- |
| **Process** | Swap the values using a temporary variable via pointers |
| **Output** | Display values before and after swapping |

CODE

#include <stdio.h>

void swap(int \*a, int \*b)

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

void main()

{

int x, y;

printf("Enter two numbers: ");

scanf("%d %d", &x, &y);

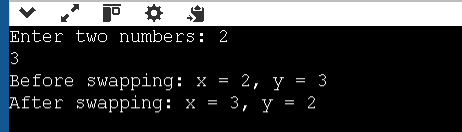
printf("Before swapping: x = %d, y = %d\n", x, y);

swap(&x, &y); // Pass addresses of x and y

printf("After swapping: x = %d, y = %d\n", x, y);

}

OUTPUT



1. Write a recursive function to find the nth Fibonacci number.

IPO

| **Input** | **An integer n representing position in Fibonacci series** |
| --- | --- |
| **Process** | Use recursion to calculate the nth Fibonacci number |
| **Output** | Fibonacci number at position n |

CODE

#include <stdio.h>

int fibonacci(int n)

{

if(n == 0)

return 0;

else if(n == 1)

return 1;

else

return fibonacci(n - 1) + fibonacci(n - 2);

}

void main()

{

int n;

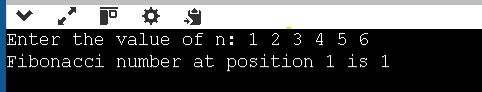
printf("Enter the value of n: ");

scanf("%d", &n);

printf("Fibonacci number at position %d is %d\n", n, fibonacci(n));

}

OUTPUT



1. Write a program to find GCD and LCM using functions.

IPO

| **Input** | **Two integers a and b** |
| --- | --- |
| **Process** | Use GCD function → Then use it to find LCM |
| **Output** | GCD and LCM of the two numbers |

CODE

#include <stdio.h>

int findGCD(int a, int b) {

int i, gcd;

for(i = 1; i <= a && i <= b; ++i) {

if(a % i == 0 && b % i == 0)

gcd = i;

}

return gcd;

}

int findLCM(int a, int b) {

int lcm;

lcm = (a \* b) / findGCD(a, b);

return lcm;

}

void main() {

int num1, num2, gcd, lcm;

printf("Enter two numbers: ");

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

gcd = findGCD(num1, num2);

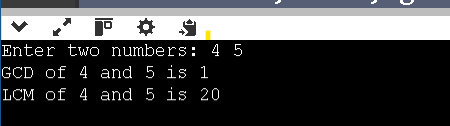
lcm = findLCM(num1, num2);

printf("GCD of %d and %d is %d\n", num1, num2, gcd);

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

}

OUTPUT



1. Write a program to demonstrate global and local variables.

IPO

| **Part** | **Description** |
| --- | --- |
| **Input** | None from the user. The program uses hardcoded variables (globalVar, localVar). |
| **Process** | 1. Declare and initialize a global variable.  2. Define local variables in main() and display() functions.  3. Print values of global and local variables. |
| **Output** | Values of global and local variables printed from both main() and display() functions. |

CODE

#include <stdio.h>

int globalVar = 100;

void display() {

int localVar = 50;

printf("Inside display function:\n");

printf("Global variable: %d\n", globalVar);

printf("Local variable: %d\n", localVar);

}

void main() {

int localMain = 25;

printf("Inside main function:\n");

printf("Global variable: %d\n", globalVar);

printf("Local variable in main: %d\n", localMain);

display();

}

OUTPUT

