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

 **Input:** The function takes a non-negative integer n.

 **Process:** It multiplies all integers from 1 to n using a loop.

 **Output:** Returns the factorial of n. For example, 5! = 5×4×3×2×1 = 120.

#include <stdio.h>

void main()

{

int number;

number = 5;

int fact = factorial(number);

if (fact == -1)

{

printf("Factorial not defined for negative numbers.\n");

}

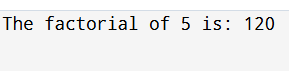
else

{

printf("The factorial of %d is: %d\n", number, fact);

}

}



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

**Input:** The function takes an integer n.

**Process:**

* If n <= 1, it's **not** prime.
* Loop from 2 to sqrt(n) (optimized) to check if any number divides n.

**Output:** Returns true if n is prime, else false.

#include <stdio.h>

int main()

{

int number = 17;

if (isPrime(number))

{

printf("%d is a prime number.\n", number);

}

Else

{

printf("%d is not a prime number.\n", number);

}

}



1. Write a function to calculate power using recursion.

**Input:** base = 2, exponent = 5

**Process:**

* 1. The function calls itself with a smaller exponent (exponent - 1) until it reaches 0.

**Output:** The result 32

#include <stdio.h>

int main()

{

int base = 2;

int exponent = 5;

int result = power(base, exponent);

printf("%d raised to the power %d is: %d\n", base, exponent, result);

}



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

**Input:** number = 121

**Process:**

* reverseNumber() recursively reverses the number.
* Then we compare reversed number with original.

**Output:** If equal → it's a palindrome.

#include <stdio.h>

int reverseNumber(int n, int rev)

{

if (n == 0)

return rev;

return reverseNumber(n / 10, rev \* 10 + n % 10);

}

int isPalindrome(int n)

{

int reversed = reverseNumber(n, 0);

return (n == reversed);

}

int main()

{

int number = 121;

if (isPalindrome(number))

printf("%d is a palindrome number.\n", number);

else

printf("%d is not a palindrome number.\n", number);

}



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

| * **Input** |
| --- |
| * Two integers n and r |

| * **Process** |
| --- |
| * Use factorial formula: nCr = n! / (r! \* (n - r)!) |

| * **Output** |
| --- |
| * Number of combinations |
| #include <stdio.h>  int factorial(int n)  {  if (n == 0 || n == 1)  return 1;  else  return n \* factorial(n - 1);  }  int nCr(int n, int r)  {  if (r > n) return 0;  return factorial(n) / (factorial(r) \* factorial(n - r));  }  int main()  {  int n = 5, r = 2;  int result = nCr(n, r);  printf("nCr(%d, %d) = %d\n", n, r, result);  } |

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

| **Input** |
| --- |
| Two integers a and b |

| **Process** |
| --- |
| Modify values using both call by value and reference |

| **Output** |
| --- |
| Demonstrate effect of each call type |

#include <stdio.h>

// Call by Value

void callByValue(int x) {

x = x + 10;

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

}

// Call by Reference

void callByReference(int \*y) {

\*y = \*y + 10;

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

}

int main() {

int a = 5, b = 5;

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

callByValue(a);

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

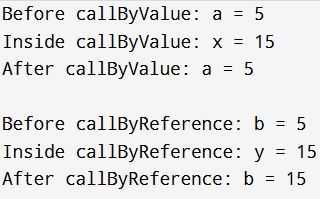
printf("Before callByReference: b = %d\n", b);

callByReference(&b);

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

return 0;

}



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

| **Input** |  | |
| --- | --- | --- |
| Two integers a, b |  | |
| **Process** | | |
| Swap values using a function | | |
|  | | |
| **Output** | | |
| a and b with swapped values | | |

#include <stdio.h>

void swap(int \*x, int \*y)

{

int temp;

temp = \*x;

\*x = \*y;

\*y = temp;

}

int main()

{

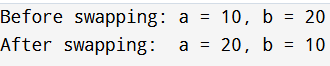
int a = 10, b = 20;

printf("Before swapping: a = %d, b = %d\n", a, b);

swap(&a, &b);

printf("After swapping: a = %d, b = %d\n", a, b);

}



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

| **Input** |  | |
| --- | --- | --- |
| An integer n (position in Fibonacci sequence) |  | |
| **Process** | | |
| Use recursion: fib(n) = fib(n-1) + fib(n-2) with base cases fib(0)=0, fib(1)=1 | | |

| **Output** |
| --- |
| The nth Fibonacci number |

#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);

}

int main()

{

int n = 7;

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

}



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

| **Input** |
| --- |
| Two integers a and b |
| **Process** |
| Use Euclidean algorithm for GCD and formula LCM = (a × b) / GCD |
| **Output** |
| GCD and LCM of the given numbers |

#include <stdio.h>

int findGCD(int a, int b)

{

if (b == 0)

return a;

else

return findGCD(b, a % b);

}

int findLCM(int a, int b)

{

return (a \* b) / findGCD(a, b);

}

int main()

{

int num1 = 12, num2 = 18;

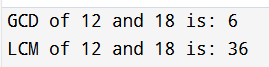
int gcd = findGCD(num1, num2);

int 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);

}



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

| **Input** |  |
| --- | --- |
| No user input |  |
| **Process** | | | |
| Use a global variable (accessible everywhere) and local variable (within a function) | | | |
| **Output** | | | |
| Shows values of global and local variables | | | |

#include <stdio.h>

int globalVar = 100;

void display()

{

int localVar = 50;

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

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

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

}

int main()

{

int localVar = 20;

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

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

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

display();

}

