**Unit 4**

**Strings**

* A string is nothing but the collection of the individual array elements or characters.
* String is enclosed within Double quotes.
* “programming" is a example of String.
* Each Character Occupy 1 byte of Memory.
* Size of “programming“ = 11 bytes
* String is always Terminated with NULL Character (‘\0′).
* char word[20] = “‘p’ , ‘r’ , ‘o’ , ‘g’ , ‘r’ , ‘a’ , ‘m’ , ‘m’ , ‘I’ , ‘n’ , ‘g’ , ‘\0’”
* NULL Character is also known as string terminating character.
* It is represented by “\0”.
* NULL Character is having ASCII value 0
* NULL terminates a string, but isn’t part of it
* important for strlen() – length doesn’t include the NULL
* Since we cannot declare string using String Data Type, instead of which we use array of type “char” to create String.

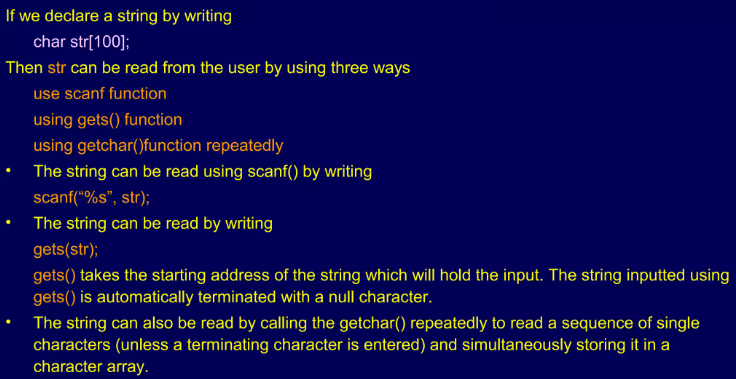
***Syntax :***

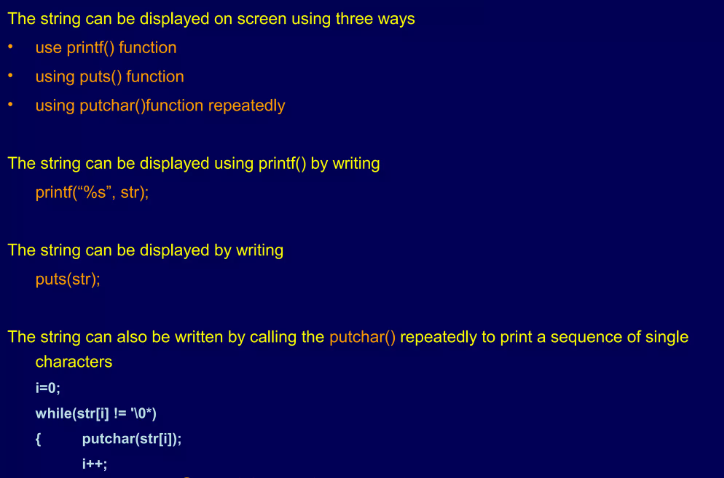
* char String\_Variable\_name [ SIZE ] ;

***Examples :***

* char city[30];
* char name[20];
* char message[50]= “Hello !! welcome”;
* String Variable must have Size specified.
* *char city[];*
* Above Statement will cause compile time error.
* When you are using string for other purpose than accepting and printing data then you must include following header file in your code– #include<string.h>
* **Initialization of String**.
* A string can be initialized in different ways. We will explain this with the help of an example.
* Below is an example to declare a string with name as str and initialize it with “HelloWorld”.

1. char str[] = “HelloWorld";
2. char str[50] = “HelloWorld";
3. char str[] = {‘H','e',’l',’l',’o',’W','o','r',’l',’d','\0'};
4. char str[11] = {‘H','e',’l',’l',’o',’W','o','r',’l',’d','\0'};





|  |
| --- |
| **Built-in String and Character Functions**  **Length, case, reverse, is digit, compare, concatenate**  #include <stdio.h>  #include <string.h>  #include <ctype.h>  int main()  {  char str[100];  printf("Enter a string: ");  scanf("%s", str); //gets(str)  // Using built-in functions  printf("Length of the string: %d\n", strlen(str););  printf("Uppercase string: %s\n", strupr(str)); // Convert to uppercase  printf("Lowercase string: %s\n", strlwr(str)); // Convert to lowercase  printf("Reversed string: %s\n", strrev(str)); // Reverse the string  // Using character functions  char ch = 'a';  printf("Is %c a digit? %s\n", ch, isdigit(ch) ? "Yes" : "No");  char str1[50], str2[50];  printf("Enter the first string: ");  scanf("%s", str1);  printf("Enter the second string: ");  scanf("%s", str2);  int result = strcmp(str1, str2);  if (result == 0)  {  printf("The strings are equal.\n");  }  else  {  printf("The strings are not equal.\n");  }  strcat(str1, str2);  printf("Concatenated string: %s\n", str1); rvce  return 0;  } |

|  |
| --- |
| #include <stdio.h>  int main ()  {  char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};  printf("Greeting message: %s\n", greeting );  return 0;  } |
| #include <stdio.h>  int main ()  {  char str[20];  printf("enter string\n");  scanf("%[aeiou]",str); // accepts only vowels  printf("%s", str);  scanf("%[^aeiou]",str); // does not accept vowels  printf("%s", str);  return 0;  } |
| **Calculate String Length without using built in function**  #include <stdio.h>  int main()  {  char str[100];  printf("Enter a string: ");  scanf("%s", str);  int i = 0;  // Iterate through the characters until null terminator is encountered  while (str[i] != '\0')  {  i++;  }  printf("Length of the string: %d\n", i);  return 0;  } |
| **Concatenate Two Strings without using built in function**  #include <stdio.h>  int main()  {  char str1[50], str2[50], result[100];  int i, j;  printf("Enter the first string: ");  scanf("%s", str1);  printf("Enter the second string: ");  scanf("%s", str2);  // Copy characters from the first string to the result string  for (i = 0; str1[i] != '\0'; ++i)  {  result[i] = str1[i];  }  // Concatenate characters from the second string to the result string  for (j = 0; str2[j] != '\0'; ++j)  {  result[i + j] = str2[j];  }  // Add null terminator to the end of the result string  result[i + j] = '\0';  printf("Concatenated string: %s\n", result);  return 0;  } |
| **Appending a String to Another String**  #include <stdio.h>  int main()  {  char str1[50] = "Hello";  char str2[] = " World";  int i, j;  // Find the end of the destination string  for (i = 0; str1[i] != '\0'; ++i);  // Append characters from the source to the destination  for (j = 0; str2[j] != '\0'; ++j, ++i)  {  str1[i] = str2[j];  }  // Add null terminator to the end of the destination string  str1[i] = '\0';  printf("Appended string: %s\n", str1);  return 0;  } |
| **Compare two Strings without using built in function**  #include <stdio.h>  main()  {  char str1[20]; // declaration of char array  char str2[20]; // declaration of char array  printf("Enter the first string : ");  scanf("%s",str1);  printf("Enter the second string : ");  scanf("%s",str2);  int flag=0,i=0; // integer variables declaration  while(str1[i]!='\0' && str2[i]!='\0') // while loop  {  if(str1[i]!=str2[i])  {  flag=1;  break;  }  i++;  }  if(flag==0)  printf("strings are same");  else  printf("strings are not same");  } |
| **Reverse a String without using built in function**  #include <stdio.h>  #include <string.h>  int main()  {  char str[100], reversedStr[100];  printf("Enter a string: ");  scanf("%s", str);  int length = strlen(str);  for (int i = 0; i < length; i++)  {  reversedStr[i] = str[length - i - 1];  }  reversedStr[length] = '\0';  printf("Reversed string: %s\n", reversedStr);  return 0;  } |
| **Check for String Palindrome**  #include <stdio.h>  #include <string.h>  int main()  {  char str[10];  int i, len, flag = 0;    printf("Enter a string: ");  scanf("%s", str);  len = strlen(str);  for (i = 0; i < len; i++)  {  if (str[i] != str[len - i - 1])  {  flag = 1;  break;  }  }  if (flag)  printf("%s is not palindrome", str);  else  printf("%s is palindrome", str);    return 0;  } |

|  |
| --- |
| **Convert Characters to Uppercase without using built in function**  #include <stdio.h>  #include <string.h>  int main() {  char str[100];  printf("Enter a string: ");  scanf("%s", str);  for (int i = 0; str[i] != '\0'; i++)  {  if (str[i] >= 'a' && str[i] <= 'z')  {  str[i] = str[i] - 'a' + 'A';  }  }  printf("Uppercase string: %s\n", str);  return 0;  } |
| **Convert Characters to Lowercase without using built in function**  **#include <stdio.h>**  **#include <string.h>**  **int main() {**  **char str[100];**  **printf("Enter a string: ");**  **scanf("%s", str);**  **for (int i = 0; str[i] != '\0'; i++)**  **{**  **if (str[i] >= 'A' && str[i] <= 'Z')**  **{**  **str[i] = str[i] - 'A' + 'a';**  **}**  **}**  **printf("Lowercase string: %s\n", str);**  **return 0;**  **}** |

**Functions:** A function is a block of code that performs a specific task. Functions provide modularity to the code by allowing you to break down the program into smaller, manageable pieces. Each function has a name, a return type, a parameter list, and a body.

[C Functions - GeeksforGeeks](https://www.geeksforgeeks.org/c-functions/)

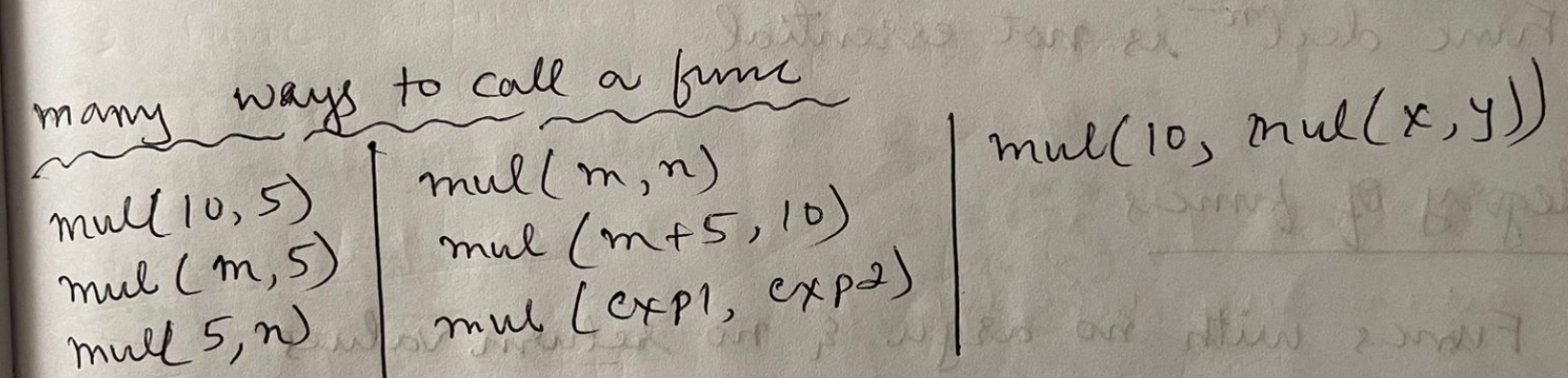
**Necessity of functions**

1. **Modularity:** Breaks down a program into smaller, manageable units.
2. **Code Reusability:** Enables reuse of code for similar tasks.
3. **Readability:** Enhances code readability and comprehension.
4. **Abstraction:** Hides implementation details, exposing only necessary information.
5. **Parameter Passing:** Allows passing values to functions for flexibility.
6. **Return Values:** Functions can produce results or perform specific operations.
7. **Debugging and Testing:** Facilitates isolated testing and debugging.
8. **Namespace Control:** Prevents naming conflicts with local variables.
9. **Code Organization:** Contributes to a well-organized program structure.
10. **Library Development:** Forms the basis for building reusable code libraries.

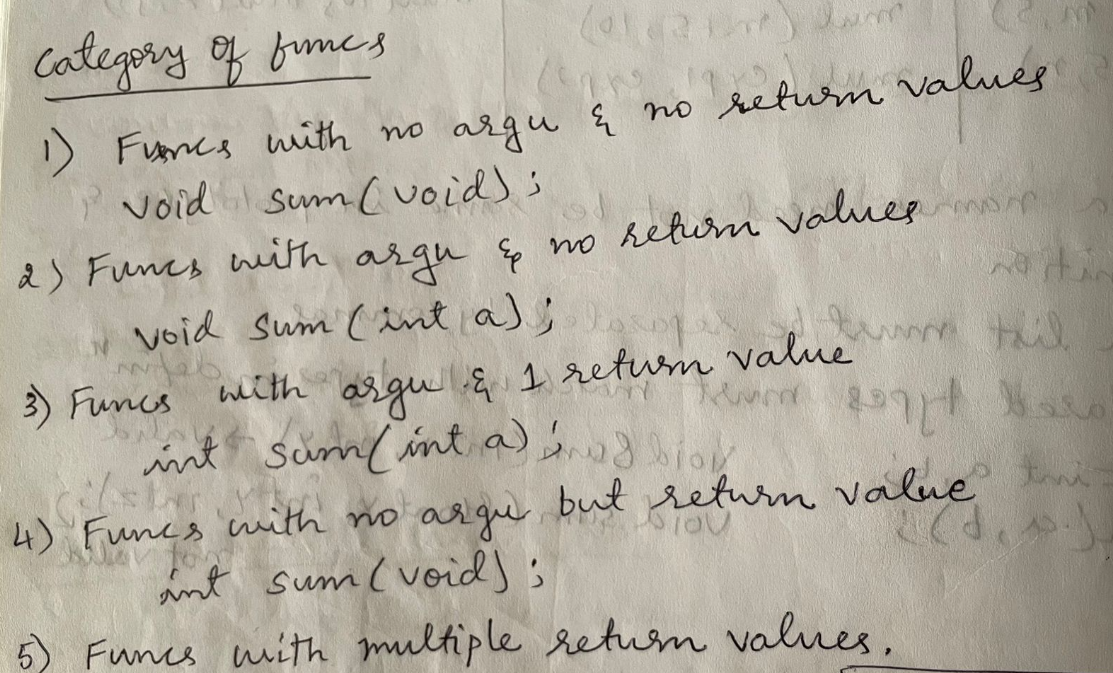
**Types of functions**

1. **Built-in Functions (Library Functions)**
2. **User-Defined Functions**

|  |
| --- |
| **Syntax of function declaration/prototype**  return\_type function\_name (parameter1\_type parameter1\_name, parameter2\_type parameter2\_name, ...);  **example:**  void sum(int x, int y); |
| **Syntax of function definition**  return\_type function\_name (parameter1\_type parameter1\_name, parameter2\_type parameter2\_name, ...)  {  // Function body  }  **example:**  void sum()  {  int a=10, b=20, c;  c=a+b;  printf(“Sum=%d”, c);  } |
| **Syntax of function call**  function\_name (parameter1\_name, parameter2\_name, ...);  **example:**  sum(a,b); |



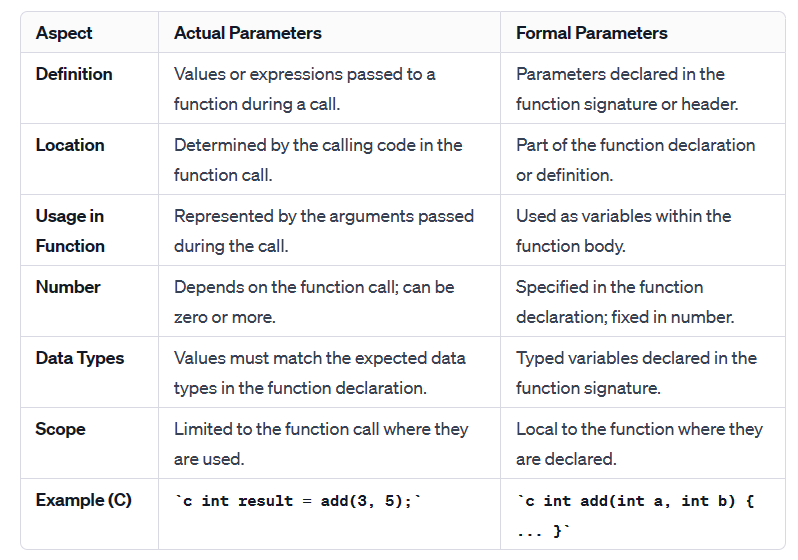
**Categories of functions**

****

|  |  |
| --- | --- |
| **Argument** | **Return value** |
| **0** | **0** |
| **0** | **1** |
| **1** | **0** |
| **1** | **1** |

|  |  |
| --- | --- |
| **Functions with no argument and no return value** | |
|  |  |
| **Functions with no arguments and return value** | |
|  |  |
| **Functions with arguments and no return value** | |
|  |  |
| **Functions with arguments and return value** | |
|  |  |

|  |
| --- |
| **Functions with no argument and no return value** |
| #include<stdio.h>  void sum();  main()  {  sum();  }  void sum()  {  int a=10, b=20, c;  c=a+b;  printf(“Sum=%d”, c);  } |
| **Functions with no arguments and return value** |
| #include<stdio.h>  int sum();  main()  {  int z;  z=sum();  printf(“Sum=%d”, z);  }  int sum()  {  int a=10, b=20, c;  c=a+b;  return c;  } |
| **Functions with arguments and no return value** |
| #include<stdio.h>  void sum(int x, int y);  main()  {  int a=10, b=20;  sum(a,b);  }  void sum(int x, int y)  {  int p;  p=x+y;  printf(“Sum=%d”, p);  } |
| **Functions with arguments and return value** |
| #include<stdio.h>  int sum(int x, int y);  main()  {  int a=10, b=20, c;  c=sum(a,b);  printf(“Sum=%d”, c);  }  int sum(int x, int y)  {  int p;  p=x+y;  return p;  } |



|  |
| --- |
| **Bubble sort**  [Introduction to Bubble Sort (youtube.com)](https://www.youtube.com/watch?v=A6m-g0SPzt0)  [Bubble Sort Algorithm | GeeksforGeeks (youtube.com)](https://www.youtube.com/watch?v=nmhjrI-aW5o)  #include<stdio.h>  #include<stdlib.h>  void bubblesort(int a[],int size);  void main()  {  int a[50],n,i;  printf("\n Enter the size of the array");  scanf("%d",&n);  printf("\n Enter the array elements: \n");  for(i=0;i<n;i++)  scanf("%d",&a[i]);  bubblesort(a,n);  printf("\n The sorted array is\n");  for(i=0;i<n;i++)  printf("%d\t",a[i]);  }  void bubblesort(int a[],int size)  {  int temp,i,j;  for(i=0;i<size;i++)  {  for(j=0;j<size-1;j++)  {  if(a[j]>a[j+1])  {  temp=a[j];  a[j]=a[j+1];  a[j+1]=temp;  }  }  }  } |
| **Linear search**  #include <stdio.h>  void linear(int a[10], int key, int n);  void linear(int a[10], int key, int n)  {  int i,flag=0;  for (i=0; i<n; i++)  {  if (key == a[i])  {  flag=1;//printf("\nkey found at position %d", i+1);  break;  }  }  if(flag==1)  printf("\nkey found at position %d", i+1);  else  printf("\nkey does not exist.");  }  int main()  {  int a[10],key,i,n;  printf("\nEnter number of elements of an array:\n");  scanf("%d",&n);  printf("\nEnter elements: \n");  for (i=0; i<n; i++)  scanf("%d", &a[i]);  printf("\nEnter key to search: ");  scanf("%d", &key);  linear(a, key, n);  return 0;  } |
| **Calculate the sum of elements in an array by passing the array to a function**  #include <stdio.h>  int sumArray(int arr[], int size)  {  int sum = 0;  for (int i = 0; i < size; i++)  {  sum += arr[i];  }  return sum;  }  int main()  {  int arr[] = {1, 2, 3, 4, 5};  int size = sizeof(arr) / sizeof(arr[0]);    int sum = sumArray(arr, size);  printf("Sum of array elements: %d\n", sum);    return 0;  } |
| **Reverse the elements of an array by passing the array to a function**  #include <stdio.h>  void reverseArray(int arr[], int size)  {  int start = 0;  int end = size - 1;  while (start < end)  {  int temp = arr[start];  arr[start] = arr[end];  arr[end] = temp;  start++;  end--;  }  }  int main()  {  int arr[] = {1, 2, 3, 4, 5};  int size = sizeof(arr) / sizeof(arr[0]);    printf("Original array: ");  for (int i = 0; i < size; i++)  {  printf("%d ", arr[i]);  }  printf("\n");    reverseArray(arr, size);    printf("Reversed array: ");  for (int i = 0; i < size; i++)  {  printf("%d ", arr[i]);  }  printf("\n");    return 0;  } |
| **Passing a 2D Array to a Function**  #include <stdio.h>  void 2DA(int arr[][3], int rows)  {  for (int i = 0; i < rows; i++)  {  for (int j = 0; j < 3; j++)  {  printf("%d ", arr[i][j]);  }  printf("\n");  }  }  int main()  {  int arr[2][3] = {{1, 2, 3}, {4, 5, 6}};    printf("2D array elements:\n");  2DA(arr, 2);    return 0;  } |
| Modify Array Elements #include <stdio.h>  void modifyArray(int arr[], int size)  {  for (int i = 0; i < size; i++)  {  arr[i] \*= 2; // Double each element  }  }  int main()  {  int arr[] = {1, 2, 3, 4, 5};  int size = sizeof(arr) / sizeof(arr[0]);    printf("Original array: ");  for (int i = 0; i < size; i++)  {  printf("%d ", arr[i]);  }  printf("\n");    modifyArray(arr, size);    printf("Modified array: ");  for (int i = 0; i < size; i++)  {  printf("%d ", arr[i]);  }  printf("\n");    return 0;  } |

**Recursion**

Recursion is the process of a function calling itself repeatedly till the given condition is satisfied. A function that calls itself directly or indirectly is called a recursive function and such kind of function calls are called recursive calls.

Recursion code is shorter than iterative code however it is difficult to understand.

Recursion cannot be applied to all the problem, but it is more useful for the tasks that can be defined in terms of similar subtasks. For Example, recursion may be applied to sorting, searching, and traversal problems.

## **Memory allocation of Recursive method**

Each recursive call creates a new copy of that method in the memory. Once some data is returned by the method, the copy is removed from the memory. Since all the variables and other stuff declared inside function get stored in the stack, therefore a separate stack is maintained at each recursive call. Once the value is returned from the corresponding function, the stack gets destroyed. Recursion involves so much complexity in resolving and tracking the values at each recursive call. Therefore we need to maintain the stack and track the values of the variables defined in the stack.

### Key Concepts of Recursion

1. **Base Case**: This is the condition under which the recursive function stops calling itself. Without a base case, the function would call itself indefinitely, leading to a stack overflow.

In recursion, a function calls itself to solve smaller instances of the same problem until it reaches the base case, which is the condition that stops the recursion. The recursive case is essential for breaking down a problem into more manageable sub-problems and making progress towards the base case.

1. **Recursive Case**: The recursive case is the portion of a recursive function that includes a call to the function itself. It typically modifies the argument(s) passed to the function in a way that reduces the problem's complexity, eventually leading to the base case.

Each time the recursive function is called, it performs some operation and then makes

another call to itself with a modified argument

**Advantages of Recursion**

* Simplifies code for problems that can naturally be divided into sub-problems.
* Easier to write and understand for problems involving hierarchical data structures, like trees and graphs.

**Disadvantages of Recursion**

* Can lead to excessive use of memory and stack overflow if not designed with care.
* Generally, less efficient in terms of execution time and memory compared to iterative solutions due to function call overhead

|  |
| --- |
| **Calculate the factorial of a number using recursion**  #include <stdio.h>  // Function to calculate factorial using recursion  int factorial(int n)  {  if (n == 0) // Base case  return 1;  else // Recursive case  return (n \* factorial(n - 1));  }  int main()  {  int num;  printf("Enter a positive integer: ");  scanf("%d", &num);  if (num < 0)  printf("Factorial is not defined for negative numbers.\n");  else  printf("Factorial of %d is %d\n", num, factorial(num));  return 0;  } |
| **Print Numbers from 1 to N using recursion**  **#**include <stdio.h>  // Function to print numbers from 1 to N  void printNumbers(int n)  {  if (n > 0) //Base case  {  printNumbers(n - 1); // Recursive case  printf("%d ", n);  }  }  int main()  {  int n;  printf("Enter a positive integer: ");  scanf("%d", &n);  printf("Numbers from 1 to %d: ", n);  printNumbers(n);  printf("\n");  return 0;  **}** |
| **Calculate sum of digits of a number using recursion**  **#**include <stdio.h>  int sumOfDigits(int n)  {  if (n == 0)  return 0;  else  return (n % 10) + sumOfDigits(n / 10);  }  int main()  {  int num;  printf("Enter a number: ");  scanf("%d", &num);  printf("Sum of digits of %d is %d\n", num, sumOfDigits(num));  return 0;  } |
| ****Power Calculation using recursion**** #include <stdio.h>  int power(int base, int exp)  {  if (exp == 0)  return 1;  else  return base \* power(base, exp - 1);  }  int main()  {  int base, exp;  printf("Enter the base and exponent: ");  scanf("%d %d", &base, &exp);  printf("%d^%d is %d\n", base, exp, power(base, exp));  return 0;  } |
| **Recursive C program to find length of a string**  #include <stdio.h>  int find\_len (char [], int);  int main ()  {  char str[100];  int len = 0;  printf ("Enter the string: \n");  scanf ("%s", str);  len = find\_len (str, 0);  printf ("The length of the given string is: %d\n", len);  return 0;  }    int find\_len (char str[], int j)  {  static int l = 0;  if (str[j] = = '\0')  return l;  else  l ++;  find\_len (str, j + 1);  }   * The **static** keyword in C, when used within a function, makes the variable retain its value across multiple calls to that function. * A **static** variable inside a function is initialized only once, and its value is preserved between function calls. |
| Print Fibonacci Series up to N using Recursion #include <stdio.h>  // Function to calculate nth Fibonacci number  int fibonacci(int n)  {  if (n <= 1)// Base case  return n;  else  return fibonacci(n - 1) + fibonacci(n - 2); // Recursive case  }  // Function to print Fibonacci series up to n terms  void printFibonacciSeries(int n) {  for (int i = 0; i < n; i++) {  printf("%d ", fibonacci(i));  }  printf("\n");  }  int main() {  int n;  printf("Enter the number of terms: ");  scanf("%d", &n);  printf("Fibonacci series up to %d terms: ", n);  printFibonacciSeries(n);  return 0;  } |