C Mathematical Functions

C Programming allows us to perform mathematical operations through the functions defined in <math.h> header file.

The <math.h> header file contains various methods for performing mathematical operations such as sqrt(), pow(), ceil(), floor() etc.

C Math Functions

There are various methods in math.h header file. The commonly used functions of math.h header file are given below.

No.	Function	Description
1)	ceil(number)	rounds up the given number. It returns the integer value which is greater than or equal to given number.
2)	floor(number)	rounds down the given number. It returns the integer value which is less than or equal to given number.
3)	sqrt(number)	returns the square root of given number.
4)	pow(base, exponent)	returns the power of given number.
5)	abs(number)	returns the absolute value of given number.

C Math Example

Let's see a simple example of math functions found in math.h header file.

```
#include<stdio.h>
#include <math.h>
int main()
{
printf("\n%f",ceil(3.6));
printf("\n%f",ceil(3.3));
printf("\n%f",floor(3.6));
printf("\n%f",floor(3.2));
printf("\n%f",sqrt(16));
printf("\n%f",sqrt(7));
printf("\n%f",pow(2,4));
printf("\n%f",pow(3,3));
printf("\n%d",abs(-12));
 return 0;
}
```

Output:

```
4.000000
```

4.000000
3.000000
3.000000
4.000000
2.645751
16.000000
27.000000
12

sin() in C

The sin() function in C is a standard library function to find the sine value of the radian angle. It is used to evaluate the trigonometric sine function of an angle. It takes the radian angle as input and returns the sin of that angle. It is defined inside <math.h> header file.

Syntax of sin()

The syntax of sin() function in C is:

```
double sin(double angle);
```

Parameters

angle: Angle in radians.

Return Value

Returns the sine of the given radian angle which ranges between +1 to -1.

Example of sin()

```
// C program to illustrate the sin() function
#include <math.h>
#include <stdio.h>

int main()
{
   double angle1 = 3.1415926;
   double angle2 = 10;

   // printing the sine value of angle1 and angle2
   printf("sin(3.14) = %.2lf\n", sin(angle1));
   printf("sin(10) = %.2lf", sin(angle2));

   return 0;
}
```

Output

sin(3.14) = 0.00
sin(10) = -0.54

cos() Function in C

cos() function in C programming is provided by the math.h header file and is used to calculate the cosine of an angle (x) where (x) is represented in radians. This function returns the cosine of a given angle value in radians for a right-angled triangle.

Syntax of cos()

```
double cos(double x);
```

Parameters of cos()

The cos(x) function takes a single parameter x, where:

x: is a double value representing the angle in radians for which the cosine value is to be computed. The value of x can range from negative to positive infinity.

Return value of cos()

The cos(x) function returns the cosine of the given angle x which is passed in the form of radians.

The below program demonstrates how we can calculate the cosine of various angles in C.

```
// C Program to calculate the cosine of various angles in C

#include <math.h>
#include <stdio.h>
int main()
{
    // 30 degrees in radians
    double a = M_PI / 6;
    // 45 degrees in radians
    double b = M_PI / 4;
    // 60 degrees in radians
    double c = M_PI / 3;
    // 90 degrees in radians
    double d = M_PI / 2;
    // Variable to store the results
    double res;

    res = cos(a);
    printf("The Cosine of %.3lf radians (30 degrees) is "
           "%.3lf\n",
           a, res);

    res = cos(b);
    printf("The Cosine of %.3lf radians (45 degrees) is "
           "%.3lf\n",
           b, res);
```

```
    res = cos(c);
    printf("The Cosine of %.3lf radians (60 degrees) is "
           "%.3lf\n",
           c, res);

    res = cos(d);
    printf("The Cosine of %.3lf radians (90 degrees) is "
           "%.3lf\n",
           d, res);

    return 0;
}
```

Output

```
The Cosine of 0.524 radians (30 degrees) is 0.866
The Cosine of 0.785 radians (45 degrees) is 0.707
The Cosine of 1.047 radians (60 degrees) is 0.500
The Cosine of 1.571 radians (90 degrees) is 0.000
```

Explanation: In the above program M_PI is a macro constant defined in the <math.h> header which represents the mathematical constant π (pi) which is approximately equal to 3.14159265358979323846

tan() Function in C

tan() function in C programming is provided by the math.h header file and it is used to calculate the tangent of an angle x where x is represented in radians. This function returns the tangent of a given angle value in radians and it only works for right-angled triangles.

Syntax of tan()

```
double tan(double x);
```

Parameters of tan()

The tan(x) function takes a single parameter x.

x: is a double value representing the angle in radians for which the tangent value is to be computed.

Return value of tan()

The tan(x) function returns the tangent of the specified angle x which is passed in the form of radian.

Example of tan() Function in C

```
// C program to calculate the tangent of 30, 45, 60 and 90 degree in C

#include <math.h>
#include <stdio.h>
```

```
int main()
{
    // 30 degrees in radians
    double a = M_PI / 6;
    // 45 degrees in radians
    double b = M_PI / 4;
    // 60 degrees in radians
    double c = M_PI / 3;
    // 90 degrees in radians
    double d = M_PI / 2;
    // Variable to store the results
    double res;

    res = tan(a);
    printf("The tangent of %.3lf radians (30 degrees) is "
           "%.3lf\n",
           a, res);

    res = tan(b);
    printf("The tangent of %.3lf radians (45 degrees) is "
           "%.3lf\n",
           b, res);

    res = tan(c);
    printf("The tangent of %.3lf radians (60 degrees) is "
           "%.3lf\n",
           c, res);

    res = tan(d);
    printf("The tangent of %.3lf radians (90 degrees) is "
           "%.3lf\n",
           d, res);

    return 0;
}
```

Output

```
The tangent of 0.524 radians (30 degrees) is 0.577
The tangent of 0.785 radians (45 degrees) is 1.000
The tangent of 1.047 radians (60 degrees) is 1.732
The tangent of 1.571 radians (90 degrees) is 16331239353195370.000
```

asin() and atan() functions in C with Example

In C++, asin() and atan() is a predefined function used for mathematical calculations. math.h is the header file required for various mathematical functions. All the functions available in this library take double as an argument and return double as the result.

asin() Method

asin() function is used to find the arc sine of a number means give a sin value to this function it will return the angle in radian corresponding to that value. In trigonometric, arc sine is the inverse operation of sine.

Syntax:

double asin(double k)

Parameters:

k is the value whose corresponding angle we have to find.

atan() Method

atan() function is used to find the arc tangent of a number means gives a tangent value to this function it will return the angle in radians corresponding to that value. arc tangent is the inverse operation of a tangent. This function accepts all the real numbers and atan() function returns the values in the range of [-?/2, ?/2].

Syntax:

double atan(double k)

Parameters:

k is the value whose corresponding angle we have to find.

Let us see the differences in a tabular form as shown below as follow:

asin()	atan()
It is used to return the principal value of the arc sine of x.	It is used to return the principal value of the arc tangent of x
It takes one parameter that is the value whose arc sine is computed	It takes one parameter that is the value whose arc tangent is computed.
Its return type is integer.	Its return type is integer.
It is defined in math.h header file.	It is defined in math.h header file.

C log() Function

log() function in C programming is a mathematical function provided by the math.h header file and is used to calculate the natural logarithm (base(e)) of a given number

passed as parameter. This function returns natural logarithm of x as double and works with double data types for other data types like float or long double, we can use logf() and logl(), respectively.

Syntax of log() in C

double log(double x);

Parameters of log()

The log(x) function takes a single parameter x, where:
x: is a value of type double whose natural logarithm is to be computed and the value of x must be positive.

Return Value of log()

The log(x) function returns the logarithm of the given number x passed as a parameter. If the value of x is negative then it returns a domain error (NaN) because the logarithm of a negative number returns an imaginary number which is not valid.

Example 1

The below program demonstrates how we can calculate the log of a given numbers using the log(x) function in C.

```
// C Program to calculate the logrithm of given numbers

#include <errno.h>
#include <math.h>
#include <stdio.h>

// Function to calculate the log of a number
void calculate_log(double value)
{
   // Reset errno before calling log
   errno = 0;
   double result = log(value);

   // Handle negative numbers
   if (errno == EDOM)
   {
      printf("Domain error: log of %f is undefined.\n",value);
   }
   else {
      printf("The natural logarithm of %f is %f\n", value,result);
   }
}

int main()
{
```

```
    // Define the numbers whose log is required
    double value1 = 5.0;
    double value2 = 10.0;
    double value3 = 15.0;
    double value4 = 20.0;
    double value5 = -10.0;

    // Calculate and print the natural logarithm for each value
    calculate_log(value1);
    calculate_log(value2);
    calculate_log(value3);
    calculate_log(value4);
    calculate_log(value5);

    return 0;
}
```

Output

```
The natural logarithm of 5.000000 is 1.609438
The natural logarithm of 10.000000 is 2.302585
The natural logarithm of 15.000000 is 2.708050
The natural logarithm of 20.000000 is 2.995732
```

Explanation: In the above program, the errno global variable is used for error handling. Whenever a negative number is provided as an parameter to the log function the value of errno is set to EDOM which denotes domain error. Domain error occurs when an input argument is outside the domain of a mathematical function.

Example 2

The below program demonstrates how we can calculate the logarithm of different data types in C.

```
// C program to calculate the logarithm of different data types
#include <math.h>
#include <stdio.h>

int main()
{
    // Declare variables of different data types
    // double type variable
    double num1 = 2.0;
    // float type variable
    float num2 = 3.0;
    // long double type variable
    long double num3 = 4.0;
```

```
    // Calculate the natural logarithm for various data types variables
    double result1 = log(num1);

    float result2 = logf(num2);

    long double result3 = logl(num3);

    // Print the results
    printf("Natural logarithm of %.2f is %.2f\n", num1, result1);
    printf("Natural logarithm of %.2f is %.2f\n", num2, result2);
    printf("Natural logarithm of %.2Lf is %.2Lf\n", num3, result3);

    return 0;
}
```

Output

```
Natural logarithm of 2.00 is 0.69
Natural logarithm of 3.00 is 1.10
Natural logarithm of 4.00 is 1.39
```

C String Functions

The C string functions are built-in functions that can be used for various operations and manipulations on strings. These string functions can be used to perform tasks such as string copy, concatenation, comparison, length, etc. The <string.h> header file contains these string functions.

In this article, we will discuss some of the most commonly used String functions in the C programming language.

String Functions in C

Some of the commonly used string functions in C are as follows:

strcat() in C

C strcat() function appends the string pointed to by src to the end of the string pointed to by dest. It will append a copy of the source string in the destination string. plus a terminating Null character. The initial character of the string(src) overwrites the Null-character present at the end of the string(dest).

It is a predefined string handling function under string library <string.h> in c and <cstring> in C++.

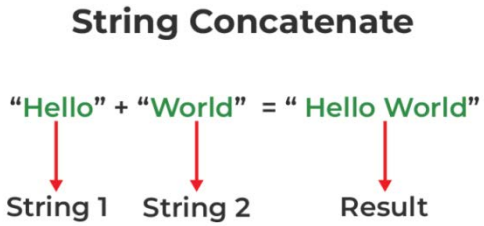
Syntax:

```
char *strcat(char *dest, const char *src);
```

Parameters: The method accepts the following parameters:
dest: This is a pointer to the destination array, which should contain a C string, and should be large enough to contain the concatenated resulting string.
src: This is the string to be appended. This should not overlap the destination.

Return value:

The strcat() function returns dest, the pointer to the destination string.



Examples:

Input: src = "ForGeeks"
dest = "Geeks"
Output: "GeeksForGeeks"

Input: src = "World"
dest = "Hello "
Output: "Hello World"

Below is the C program to implement the above approach

```
// C program to implement the above approach
#include <stdio.h>
#include <string.h>

// Driver code
int main()
{
   // Define a temporary variable
   char example[100];

   // Copy the first string into the variable
   strcpy(example, "Geeks");

   // Concatenate this string to the end of the
first one
   strcat(example, "ForGeeks");

   // Display the concatenated strings
   printf("%s\n", example);

   return 0;
}
```

Output

GeeksForGeeks

The behavior of strcat() is undefined if:
the destination array is not large enough for the contents of both src and dest and the terminating null character
if the string overlaps.
if either dest or src is not a pointer to a null-terminated byte string.

strlen() function in c

The strlen() function in C calculates the length of a given string. The strlen() function is defined in string.h header file. It doesn't count the null character '\0'.

Syntax of C strlen()

The syntax of strlen() function in C is as follows:

```
size_t strlen(const char* str);
```

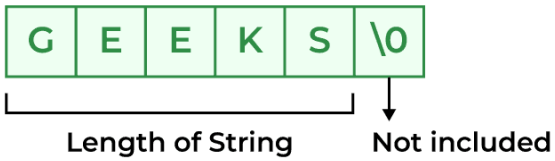
Parameters

The strlen() function only takes a single parameter.

str: It represents the string variable whose length we have to find.

Return Value

This function returns the integral length of the string passed.



C strlen() Function

Example of C strlen()

The below programs illustrate the strlen() function

```
// c program to demonstrate
// example of strlen() function.
#include <stdio.h>
#include <string.h>

int main()
{
   // defining string
   char str[] = "GeeksforGeeks";

   // getting length of str using strlen()
   int length = strlen(str);
   printf("Length of string is : %d",
length);
```

```
    return 0;
}
```

Output
Length of string is : 13

Important Points about strlen()

The following points should be kept in mind while using strlen():
strlen() does not count the NULL character '\0'.
The time complexity of strlen() is O(n), where n is the number of characters in the string.
Its return type is size_t (which is generally unsigned int).

C strcmp()

In C language, the <string.h> header file contains the Standard String Library that contains some useful and commonly used string manipulation functions. In this article, we will see how to compare strings in C using the function strcmp().

What is strcmp() in C?

C strcmp() is a built-in library function that is used for string comparison. This function takes two strings (array of characters) as arguments, compares these two strings lexicographically, and then returns 0,1, or -1 as the result. It is defined inside <string.h> header file with its prototype as follows:

Syntax of strcmp() in C

```
strcmp(first_str, second_str );
```

Parameters of strcmp() in C

This function takes two strings (array of characters) as parameters:
first_str: First string is taken as a pointer to the constant character (i.e. immutable string).
second_str: Second string is taken as a pointer to a constant character.

The strcmp() function returns three different values after the comparison of the two strings which are as follows:

1. Zero (0)

A value equal to zero when both strings are found to be identical. That is, all of the characters in both strings are the same.

2. Greater than Zero (> 0)

A value greater than zero is returned when the first not-matching character in first_str has a greater ASCII value than the corresponding character in second_str or we can also say that if the character in first_str is lexicographically after the character of second_str, then zero is returned.

3. Lesser than Zero (< 0)

A value less than zero is returned when the first not-matching character in first_str has a lesser ASCII value than the corresponding character in second_str. We can also say that if the character in first_str is lexicographically before the character of second_str, zero is returned.

How to use the strcmp() function in C

The following example demonstrates how to use the strcmp() function in C

```
// C Program to Demonstrate the use of strcmp() function
#include <stdio.h>
#include <string.h>

int main()
{
   // declaring two same string
   char* first_str = "Geeks";
   char* second_str = "Geeks";

   // printing the strings
   printf("First String: %s\n", first_str);
   printf("Second String: %s\n", second_str);

   // printing the return value of the strcmp()
   printf("Return value of strcmp(): %d",
         strcmp(first_str, second_str));

   return 0;
}
```

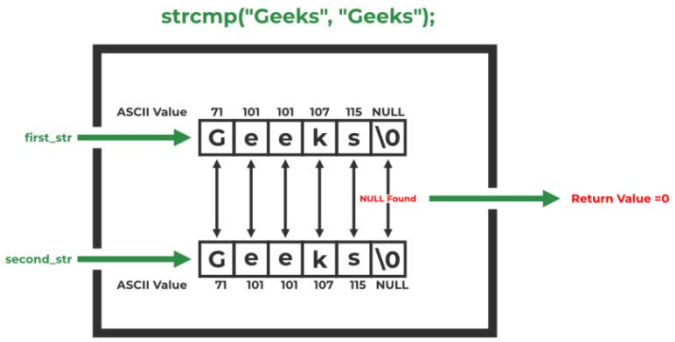
Output

First String: Geeks
Second String: Geeks

How strcmp() in C works?

C strcmp() function works by comparing the two strings lexicographically. It means that it compares the ASCII value of each character till the non-matching value is found or the NULL character is found. The working of the C strcmp() function can be described as follows:

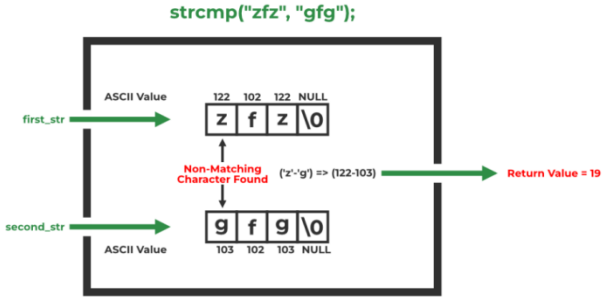
1. It starts with comparing the ASCII values of the first characters of both strings.
2. If the first characters in both strings are equal, then this function will check the second character, if they are also equal, then it will check the third, and so on till the first unmatched character is found or the NULL character is found.
3. If a NULL character is found, the function returns zero as both strings will be the same.



4. If a non-matching character is found,

If the ASCII value of the character of the first string is greater than that of the second string, then the positive difference (> 0) between their ASCII values is returned.

If the ASCII value of the character of the first string is less than that of the second string, then the negative difference (< 0) between their ASCII values is returned.



All of these three cases are demonstrated in the below examples.

Examples of strcmp() in C

Example 1. strcmp() behavior for identical strings

This program illustrates the behavior of the strcmp() function for identical strings.

```
// C program to illustrate
// strcmp() function
#include<stdio.h>
#include<string.h>

int main()
{

   char first_str[] = "g f g";
   char second_str[] = "g f g";

   // Using strcmp()
   int res = strcmp(first_str, second_str);

   if (res==0)
      printf("Strings are equal");
   else
      printf("Strings are unequal");

   printf("\nValue returned by strcmp() is: %d" , res);
   return 0;
}
```

Output

```
Strings are equal
Value returned by strcmp() is: 0
```

Example 2. strcmp() behavior for the lexicographically greater first string

The below example demonstrates the strcmp() function behavior for the lexicographically greater first string.

```
// C program to illustrate strcmp() function

#include<stdio.h>
#include<string.h>
int main()
{
   // z has greater ASCII value than g
   char first_str[] = "zfz";
   char second_str[] = "gfg";

   int res = strcmp(first_str, second_str);

   if (res==0)
      printf("Strings are equal");
   else
      printf("Strings are unequal");

   printf("\nValue of result: %d" , res);

   return 0;
}
```

Output

```
Strings are unequal
Value of result: 19
```

Example 3. strcmp() behavior for the lexicographically smaller first string.

The below example demonstrates the strcmp() function behavior for the lexicographically smaller first string.

```
// C program to illustrate strcmp() function
#include<stdio.h>
#include<string.h>
int main()
{
   // b has less ASCII value than g
   char first_str[] = "bfb";
   char second_str[] = "gfg";

   int res = strcmp(first_str, second_str);

   if (res==0)
```

```
        printf("Strings are equal");
    else
        printf("Strings are unequal");

    printf("\nValue returned by strcmp() is: %d" , res);


    return 0;
}
```

Output
Strings are unequal
Value returned by strcmp() is: -5

strcpy in C

strcpy is a C standard library function that copies a string from one location to another. It is defined in the string.h header file.

The function takes two arguments: a destination buffer where the copied string will be stored, and a source string that will be copied. The function copies the entire source string, including the null terminator, into the destination buffer.

The C strcpy() function copies the content of a string to another. The content of the destination string will be replaced with that of the source string by the strcpy() function. It is defined inside <string.h> header file.

Syntax:

```
char* strcpy(char* destination, const char* source);
```

Parameters: This method accepts the following parameters:
destination: Pointer to the destination character array where the content is to be copied.
source: Pointer to the source character array which is to be copied.

Return Value: A pointer to the destination string is returned after the strcpy() function copies the source string.

Example: 1

```
// C program to illustrate
// strcpy() function in C
#include <stdio.h>
#include <string.h>

int main()
{
    char str1[] = "Hello World!";
    char str2[] = "GfG";
```

```
    char str3[40];
    char str4[40];
    char str5[] = "GeeksForGeeks";

    strcpy(str2, str1);
    strcpy(str3, "Copy successful");
    strcpy(str4, str5);
    printf("str1: %s\nstr2: %s\nstr3: %s\nstr4:%s\n",str1,str2,str3,str4);
    return 0;
}
```

Output

```
str1: Hello World!
str2: Hello World!
str3: Copy successful
str4:GeeksForGeeks
```

EXAMPLE 2 :

```
#include <stdio.h>
#include <string.h>

int main()
{
    char str1[20] = "Hello";
    char str2[20];

    strcpy(str2, str1);

    printf("str1: %s\n", str1);
    printf("str2: %s\n", str2);

    return 0;
}
```

Output

```
str1: Hello
str2: Hello
```

Important Points

Using this function, you can copy the entire string to the destination string. Source strings are not appended to destination strings. As a result, the content of the destination string is replaced by the content of the source string.

Source strings are not affected. After copying, the source string remains the same.

To use strcpy(), the string.h header file must be included.

In the case of a longer source string (Character Array), strcpy() performs undefined behavior.

Some advantages of using strcpy in C include:

- It is a simple and easy-to-use function that can be used to copy strings quickly and easily.
- It is a standard library function, so it is widely available and portable across different platforms and compilers.
- It is relatively fast, as it only requires a single pass through the source string to copy it.

However, there are also some disadvantages to consider when using strcpy:

It does not check the size of the destination buffer, so it is possible to overwrite the buffer and cause a buffer overflow if the source string is longer than the destination buffer. This can lead to security vulnerabilities and other problems.

It does not handle overlapping strings properly. If the source and destination strings overlap, the behavior of strcpy is undefined.

It does not handle null characters within the source string properly. If the source string contains a null character, strcpy will stop copying at that point, even if there are additional characters in the source string.

strlwr() function in C

The strlwr( ) function is a built-in function in C and is used to convert a given string into lowercase.

Syntax:

```
char *strlwr(char *str);
```

Parameter:

str: This represents the given string which we want to convert into lowercase.

Returns: It returns the modified string obtained after converting the characters of the given string str to lowercase.

Note: This is a non-standard function that works only with older versions of Microsoft C.

Below programs illustrate the strlwr() function in C:

Example 1:-

```
// C program to demonstrate example of strlwr() function

#include<stdio.h>
#include<string.h>

int main()
{
   char str[ ] = "GEEKSFORGEEKS IS THE BEST";
```

```
    // converting the given string into lowercase.
    printf("%s\n",strlwr (str));

    return  0;
}
```

Output:
geeksforgeeks is the best

Example 2:-

```
// C program to demonstrate example of strlwr() function.

#include<stdio.h>
#include <string.h>

int main()
{
  char str[] = "CompuTer ScienCe PoRTAl fOr geeKS";

  printf("Given string is: %s\n",str);

  printf("\nString after converting to the lowercase is: %s",strlwr(str));

  return 0;
}
```

Output:
Given string is: CompuTer ScienCe PoRTAl fOr geeKS

String after converting to the lowercase is: computer science portal for geeks

strupr() function in c

The strupr( ) function is used to converts a given string to uppercase.

Syntax:

```
char *strupr(char *str);
```

Parameter:
str: This represents the given string which we want to convert into uppercase.
Returns: It returns the modified string obtained after converting the characters of the given string str to uppercase.

Below programs illustrate the strupr() function in C:

Example 1:-

```
// c program to demonstrate example of strupr() function.
#include<stdio.h>
#include<string.h>

int main()
{
   char str[ ] = "geeksforgeeks is the best";
   //converting the given string into uppercase.
   printf("%s\n", strupr (str));
   return 0;
}
```

Output:
GEEKSFORGEEKS IS THE BEST

Example 2:-

```
// c program to demonstrate example of strupr() function.

#include<stdio.h>
#include <string.h>

int main()
{
char str[] = "CompuTer ScienCe PoRTAl fOr geeKS";

printf("Given string is: %s\n", str);

printf("\nstring after converting to the uppercase is: %s", strupr(str));
return 0;
}
```

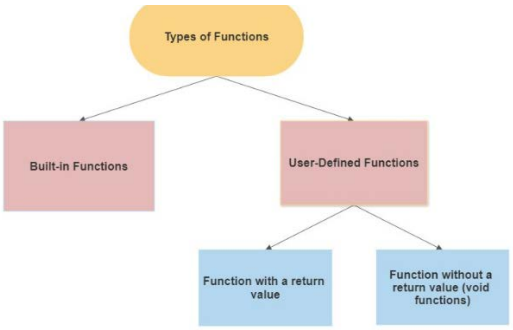
Output:
Given string is: CompuTer ScienCe PoRTAl fOr geeKS

string after converting to the uppercase is: COMPUTER SCIENCE PORTAL FOR GEEKS

What are Functions in C?

Functions are a fundamental building block of C programs and are used to modularize and organize code. They provide a way to break down a program into smaller, manageable units, which makes the code more readable, reusable, and maintainable.

They are of two types: Built-in and User Defined.



1. Built-in Functions

Built-in functions in C, often referred to as library functions or standard library functions, are predefined functions that are part of the C standard library. These functions provide essential capabilities to C programmers and can be used without the need for writing their implementations.

2. User Defined Functions

User-defined functions in programming are functions created by the programmer to perform specific tasks or operations within a program. These functions are not predefined or built into the programming language but are written by the programmer to modularize code, improve code organization, enhance reusability, and make the program more understandable.

Parts of a User-Defined Function

Function Return Type: The type of value that the function returns (e.g., int, float, void for no return value).

Function Name: The name assigned to the function, which is used to call it in the program.

Parameters: The inputs to the function, are specified within parentheses and separated by commas (optional).

Function Body: The block of code which constitutes the function and contains statements that define what the function does.

Return Statement: A statement which returns a value from the function to the calling code (optional, depending on return type).

Syntax of a User-Defined Function in C

```
return_type function_name(parameter_list)
{
   // Function body
  // ...
  return value; // Optional, based on return type
}
```

For Example

```
#include <stdio.h>

// Function Declaration
int add(int a, int b);

int main()
{
   int result = add(8, 9);
```

```
    printf("The sum is a + b = %d", result);
    return 0;
}

// Function Definition
int add(int a, int b)
{
    return a + b;
}
```

Output

The sum is a + b = 17

In the above code, a function named add is defined, which takes two integer parameters and returns their sum. The add function is declared before main, and defined after main. Inside main, the add function is called with two arguments, and its return value is printed.

Different Types of User-defined Functions in C

There are four types of user-defined functions divided on the basis of arguments they accept and the value they return:

- Function with no arguments and no return value
- Function with no arguments and a return value
- Function with arguments and no return value
- Function with arguments and with return value

1. Function with No Arguments and No Return Value

Functions that have no arguments and no return values. Such functions can either be used to display information or to perform any task on global variables.

Syntax

```
// void return type with no arguments
void function_name()
{
    // no return value
    return;
}
```

Example

```
// C program to use function with no argument and no return values
#include <stdio.h>

void sum()
```

```
{
    int x, y;
    printf("Enter x and y\n");
    scanf("%d %d", &x, &y);
    printf("Sum of %d and %d is: %d", x, y, x + y);
}

// Driver code
int main()
{
    // function call
    sum();

    return 0;
}
```

Output

Enter x and y
Sum of 4195522 and 0 is: 4195522

Explanation

In the above program, function sum does not take any arguments and has no return values. It takes x and y as inputs from the user and prints them inside the void function.

2. Function with No Arguments and With Return Value

Functions that have no arguments but have some return values. Such functions are used to perform specific operations and return their value.

Syntax

```
return_type function_name()
{
  // program
  return value;
}
```

Example

```
// C program to use function with no argument and with return
```

values

```
#include <stdio.h>

int sum()
{
   int x, y, s = 0;
   printf("Enter x and y\n");

   scanf("%d %d", &x, &y);
   s = x + y;
   return s;
}

// Driver code
int main()
{
   // function call
   printf("Sum of x and y is %d", sum());
   return 0;
}
```

Output

Enter x and y
Sum of x and y is 4195536

Explanation

In the above program, function sum does not take any arguments and has a return value as an integer type. It takes x and y as inputs from the user and returns them.

3. Function With Arguments and No Return Value

Functions that have arguments but no return values. Such functions are used to display or perform some operations on given arguments.

Syntax

```
void function_name(type1 argument1, type2 argument2,...typeN argumentN)
{
   // Program
   return;
}
```

Example

```
// C program to use function with argument and no return values
#include <stdio.h>

void sum(int x, int y)
{
   printf("Sum of %d and %d is: %d", x, y, x + y);
}

// Driver code
int main()
{
   int x, y;
   printf("Enter x and y\n");

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

   // function call
   sum(x, y);

   return 0;
}
```

Output

Enter x and y
Sum of 0 and 0 is: 0

Explanation

In the above program, function sum does take x and y as arguments and has no return value. The main function takes x and y as inputs from the user and calls the sum function to perform the print operation on the given arguments.

4. Function With Arguments and With Return Value

Functions that have arguments and some return value. These functions are used to perform specific operations on the given arguments and return their values to the user.

Syntax

```
return_type function_name(type1 argument1, type2 argument2,...typeN argumentN)
{
   // program
   return value;
}
```

Example

```
// C program to use function with argument and with return values
#include <stdio.h>

int sum(int x, int y) { return x + y; }

// Driver code
int main()
{
   int x, y;
   printf("Enter x and y\n");
   scanf("%d %d", &x, &y);

   // function call
   printf("Sum of %d and %d is: %d", x, y, sum(x, y));

   return 0;
}
```

Output

```
Enter x and y
Sum of 0 and 0 is: 0
```

Explanation

In the above program, function sum takes two arguments as x and y, and has a return value as an integer type. The main function takes input x and y from the user and calls the sum function to perform a specific operation on the given arguments and returns the value.