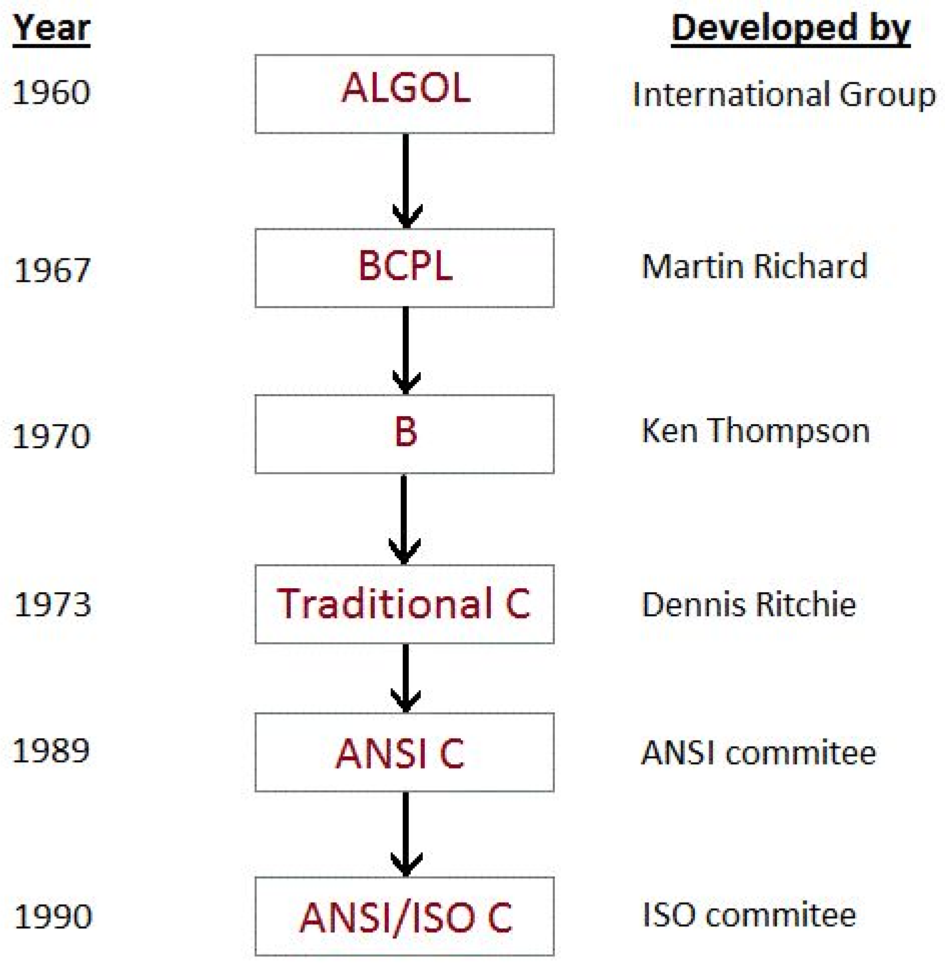
Overview of C Language

C is a structured programming language developed by Dennis Ritchie in 1973 at Bell Laboratories. It is one of the most popular computer languages today because of its structure, high-level abstraction, machine independent feature etc. C language was developed to write the UNIX operating system, hence it is strongly associated with UNIX, which is one of the most popular network operating system in use today and heart of internet data superhighway.

History of C language

C language has evolved from three different structured language ALGOL, BCPL and B Language. It uses many concepts from these languages while introduced many new concepts such as datatypes, struct, pointer etc. In 1988, the language was formalised by **American National Standard Institute**(ANSI). In 1990, a version of C language was approved by the **International Standard Organisation**(ISO) and that version of C is also referred to as C89.



The idea behind creating C language was to create an easy language which requires a simple compiler and enables programmers to efficiently interact with the machine/system, just like machine instructions.

C language compiler converts the readable C language program into machine instructions.

Importances/ Advantages of C

- Robust Language:
  It is a robust language with rich set of built-in functions and operators that can be used to write any complex program.

- Efficient and Fast
  Programs Written in C are efficient and fast. This is due to its variety of data type and powerful operators.

- Structured Language
  C is a structured language as it has a fixed structure. A program can be divided into a number of modules or blocks. A collection of these modules makes a complete program.

- Extendibility
  A C program may contain a number of user defined functions. We can add our own user defined functions to the C library if required.

- Portable
  C is highly portable this means that programs once written can be run on another machines with little or no modification.

- Rich System Library
  There are large numbers of built in functions, keywords and operators in C's System library organized in different header files. Using these built-in functions saves our time and effort.

First C program and its Structure

Lets see how to write a simple and most basic C program:

```
#include <stdio.h>
int main()
{
   printf("Hello World");  //single line comment
   return 0;
  /* multi line  comments /*
}
```

Output:

```
Hello world
```

Different parts of C program

- Pre-processor
- Header file
- Function
- Variables
- Statements & expressions
- Comments

All these are essential parts of a C language program.

Pre-processor

#include is the first word of any C program. It is also known as a **pre-processor**. The task of a pre-processor is to initialize the environment of the program, i.e to link the program with the header files required.

So, when we say #include <stdio.h>, it is to inform the compiler to include the **stdio.h** header file to the program before executing it.

Header file

A Header file is a collection of built-in(readymade) functions, which we can directly use in our program. Header files contain definitions of the functions which can be incorporated into any C program by using pre-processor #include statement with the header file.

For example, to use the printf() function in a program, which is used to display anything on the screen, the line #include <stdio.h> is required because the header file **stdio.h** contains the printf() function.

Function

Functions are main building blocks of any C Program. Every C Program will have one or more functions and there is one mandatory function which is called main()function. This function is prefixed with keyword int which means this function returns an integer value when it exits. This integer value is returned using return statement.

The C Programming language provides a set of built-in functions. printf()is a C built-in function which is used to print anything on the screen.

Variables:

Variables are used to hold numbers, strings and complex data for manipulation.

Statements & Expressions:

Expressions combine variables and constants to create new values. Statements are expressions, assignments, function calls, or control flow statements which make up C programs.

Comments

We can add comments in our program to describe what we are doing in the program. These comments are ignored by the compiler and are not executed.

To add a single line comment, start it by adding two forward slashes // followed by the comment.

To add multiline comment, enclose it between /* .... */, just like in the program above.

Return statement - return 0;

A return statement is just meant to define the end of any C program.

Data Types

Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

There are four data types in C language. They are,

Types	Data Types
Basic Data Type	int, char, float, double
Enumerated Data Type	enum
Derived data type	pointer, array, structure, union
Void data type	void

Basic Data Type

There are several basic data types in C. They are as discussed below;

1. Integer Data Type :

    - Integer data type allows a variable to store numeric values.
    - “int” keyword is used to refer integer data type.
    - The storage size of int data type is 2 or 4 or 8 byte.
    - It varies depend upon the processor in the CPU that we use. If we are using 16 bit processor, 2 byte (16 bit) of memory will be allocated for int data type.
    - Like wise, 4 byte (32 bit) of memory for 32 bit processor and 8 byte (64 bit) of memory for 64 bit processor is allocated for int datatype.
    - int (2 byte) can store values from -32,768 to +32,767
    - int (4 byte) can store values from -2,147,483,648 to +2,147,483,647.
    - If you want to use the integer value that crosses the above limit, you can go for “long int” and “long long int” for which the limits are very high.

2. Character Data Type :

    - Character data type allows a variable to store only one character.
    - Storage size of character data type is 1. We can store only one character using character data type.
    - “char” keyword is used to refer character data type.
    - For example, ‘A’ can be stored using char datatype. You can’t store more than one character using char data type.

3. Floating Point Data Type :

Floating point data type consists of 2 types. They are,

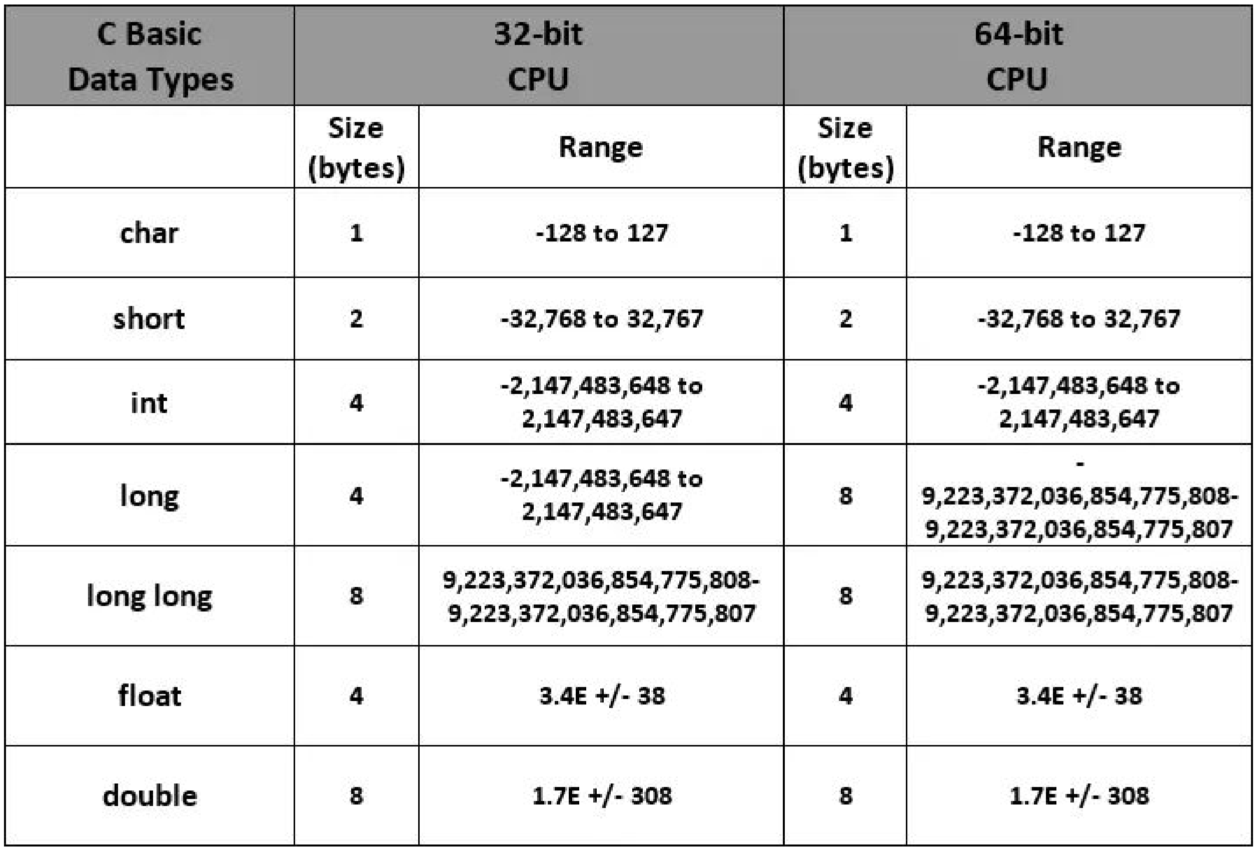
- float
- double

a) . float:

- Float data type allows a variable to store decimal values.
- Storage size of float data type is 4 byte.
- We can use up-to 6 digits after decimal using float data type.
- The range for float data type is from 3.4 E-38 to 3.4E +38.
- For example, 10.456789 can be stored in a variable using float data type.

b). double:

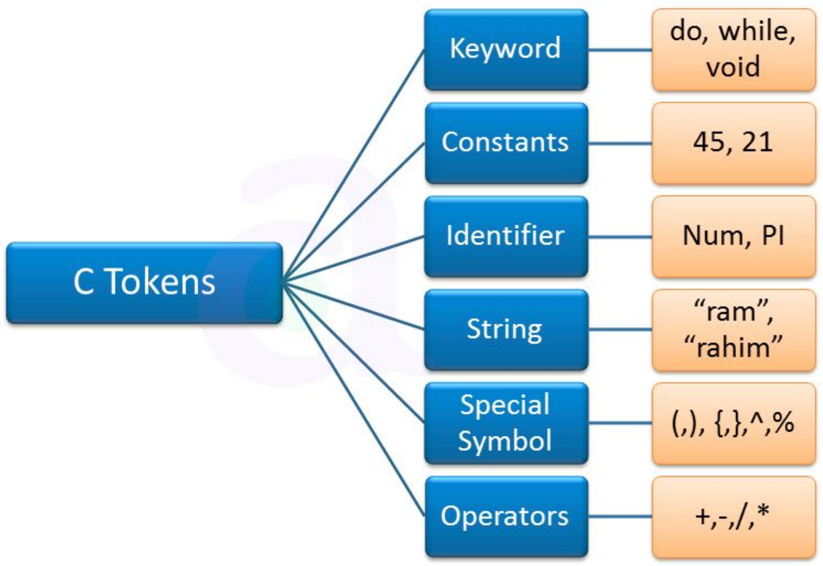
- Double data type is also same as float data type which allows up-to 10 digits after decimal.
- Storage size of double float data type is 4 byte.
- The range for double datatype is from 1.7E-308 to 1.7E+308.



C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	- 9,223,372,036,854,775,808- 9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308

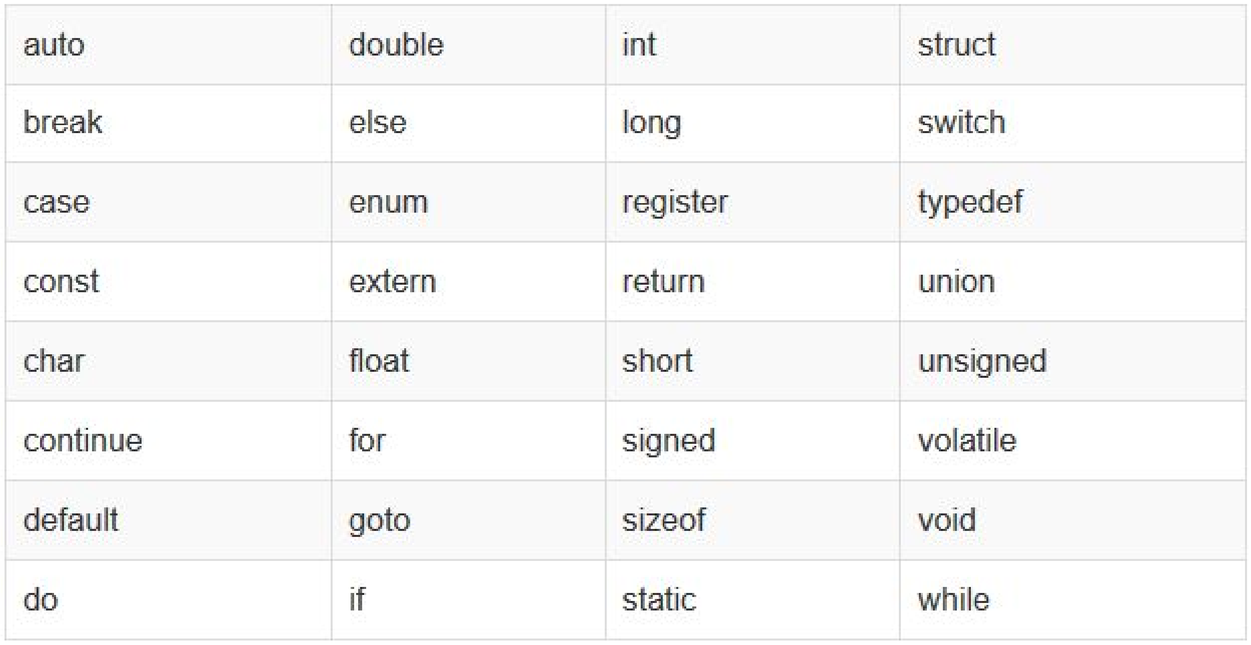
C Tokens

C tokens are the basic buildings blocks in C language which are constructed together to write a C program. Each and every smallest individual units in a C program are known as C tokens. C tokens are of six types:



Keywords

Keywords are preserved words that have special meaning in C language. The meaning of C language keywords has already been described to the C compiler. These meaning cannot be changed. Thus, keywords cannot be used as variable names because that would try to change the existing meaning of the keyword, which is not allowed. There are a total 32 keywords in C language.



auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
const	extern	return	union
char	float	short	unsigned
continue	for	signed	volatile
default	goto	sizeof	void
do	if	static	while

Identifiers:

In C language identifiers are the names given to variables, constants, functions and user-defined data. These identifiers are defined against a set of rules.

Rules for an Identifier

1. An Identifier can only have alphanumeric characters(a-z , A-Z , 0-9) and underscore(_).
2. The first character of an identifier can only contain alphabet(a-z , A-Z) or underscore (_).
3. Identifiers are also case sensitive in C. For example **name** and **Name** are two different identifiers in C.
4. Keywords are not allowed to be used as Identifiers.
5. No special characters, such as semicolon, period, whitespaces, slash or comma are permitted to be used in or as Identifier.

Constants

Constants in C are the fixed values that are used in a program, and its value remains the same during the entire execution of the program. They are also known as literals.

Constants are categorized into two basic types, and each of these types has its subtypes/categories.

1. Numeric Constants
    - Integer Constants
    - Real Constants
2. Character Constants
    - Single Character Constants
    - String Constants
    - Backslash Character Constants

Integer Constant

It's referring to a sequence of digits. Integers are of three types viz:

1. Decimal Integer
2. Octal Integer

3. Hexadecimal Integer

Example: 15, -265, 0, 99818, +25, 045, 0X6

Real Constant

The numbers containing fractional parts like 99.25 are called real or floating points constant.

Single Character Constant

It simply contains a single character enclosed within ' and ' (a pair of single quote). It is to be noted that the character '**8**' is not the same as **8**. Character constants have a specific set of integer values known as ASCII values (American Standard Code for Information Interchange).

Example: 'X', '5', ';'

String Constant

These are a sequence of characters enclosed in double quotes, and they may include letters, digits, special characters, and blank spaces. It is again to be noted that "**G**" and '**G**' are different - because "G" represents a string as it is enclosed within a pair of double quotes whereas 'G' represents a single character.

Examples: "Hello!", "2015", "2+1"

Backslash Character constant

C supports some character constants having a backslash in front of it. The lists of backslash characters have a specific meaning which is known to the compiler. They are also termed as "Escape Sequence".

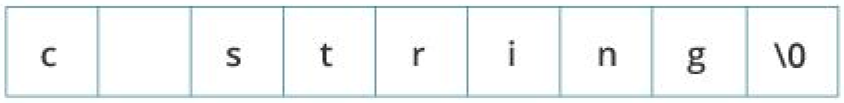
Example: \n is used for new line .

String in C

In C programming, a string is a sequence of characters terminated with a null character \0. For example:

```c
char c[] = "c string";
```

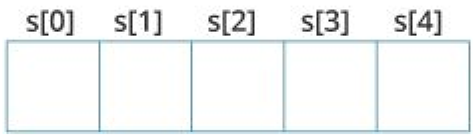
When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default.



<u>Declaration:</u>

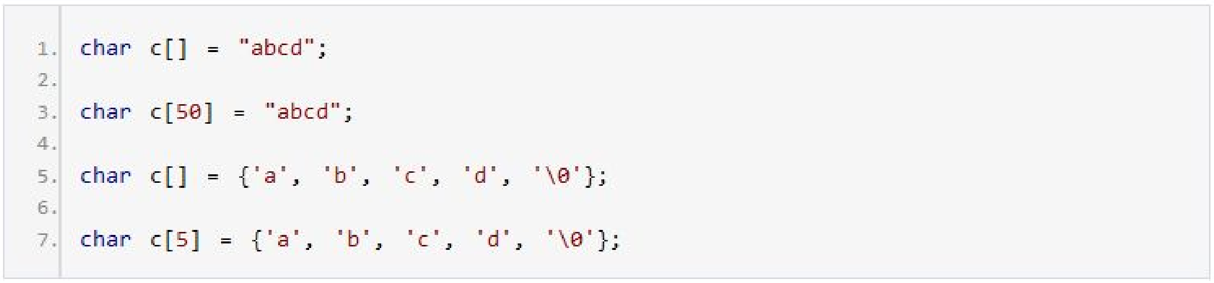
String can be declared as:

```c
char s[5];
```

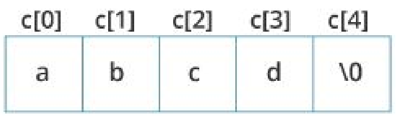


<u>Initialization:</u>

You can initialize strings in a number of ways.



```c
char c[] = "abcd";

char c[50] = "abcd";

char c[] = {'a', 'b', 'c', 'd', '\0'};

char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

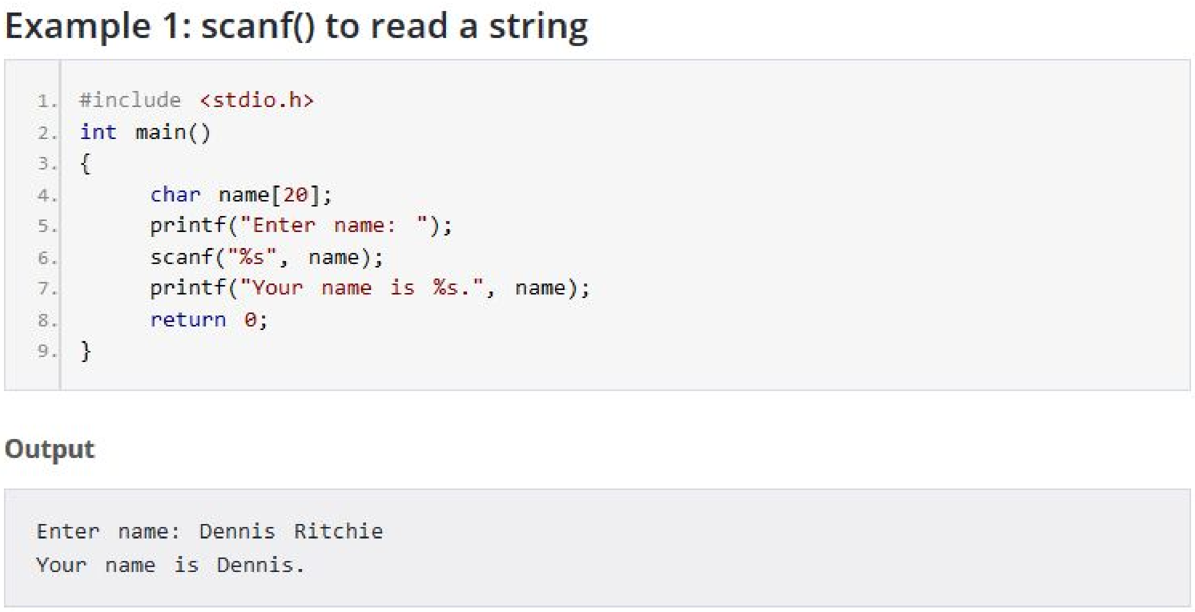


c[0]	c[1]	c[2]	c[3]	c[4]
a	b	c	d	\0

Reading String from User

You can use the scanf() function to read a string.

The scanf() function reads the sequence of characters until it encounters whitespace (space, newline, tab etc.).



Example 1: scanf() to read a string

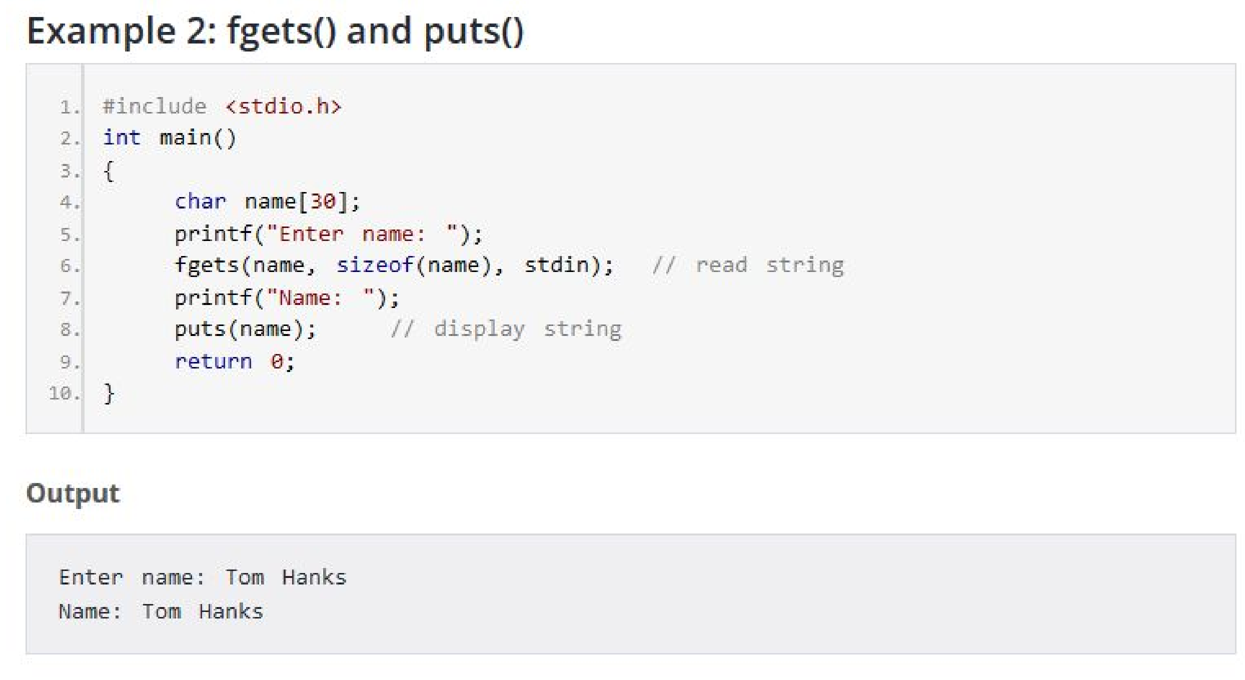
```c
#include <stdio.h>
int main()
{
     char name[20];
     printf("Enter name: ");
     scanf("%s", name);
     printf("Your name is %s.", name);
     return 0;
}
```

Output

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

Even though Dennis Ritchie was entered in the above program, only "Dennis" was stored in the name string. It's because there was a space after Dennis.

Note:You can use the fgets() function to read a line of string. And, you can use puts() to display the string.



Example 2: fgets() and puts()

```c
#include <stdio.h>
int main()
{
      char name[30];
      printf("Enter name: ");
      fgets(name, sizeof(name), stdin);  // read string
      printf("Name: ");
      puts(name);    // display string
      return 0;
}
```

Output

```
Enter name: Tom Hanks
Name: Tom Hanks
```

Here, we have used fgets() function to read a string from the user.

fgets(name, sizeof(name), stdlin); // read string

The sizeof(name) results to 30. Hence, we can take a maximum of 30 characters as input which is the size of the name string.

To print the string, we have used puts(name);.

Note: The gets() function can also be to take input from the user. However, it is removed from the C standard.

It's because gets() allows you to input any length of characters. Hence, there might be a buffer overflow.

Commonly used C String Functions:

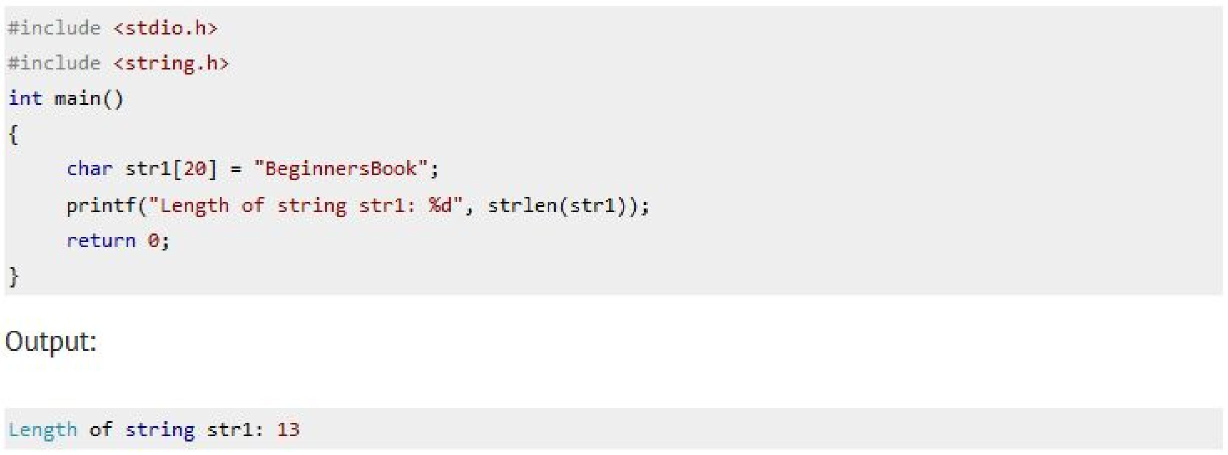
There are several string functions provided by C. These string functions are defined in string.h header file. Therefore, we need to include string.h header file using #include<string.h> at the top of the program.

Commonly used string functions in C are:

- **strlen()**- Gives the length of string
- **strcat()**- Concatenates string at the end of another string.
- **strcmp()**- Compares two strings.
    - Returns 0 if str1 is same as str2.
    - Returns <0 if strl < str2.
    - Returns >0 if str1 > str2
- **strcpy()**- Copies one string to another.
- **strupr()**- Converts string to uppercase.
- **strlwr()**- Converts string to lowercase.

Examples:

1. Example of strlen() function



```
#include <stdio.h>
#include <string.h>
int main()
{
     char str1[20] = "BeginnersBook";
     printf("Length of string str1: %d", strlen(str1));
     return 0;
}
```

Output:

```
Length of string str1: 13
```

2. Example of strcat() function

```
#include <stdio.h>
#include <string.h>
int main()
{
     char s1[10] = "Hello";
     char s2[10] = "World";
     strcat(s1,s2);
     printf("Output string after concatenation: %s", s1);
     return 0;
}
```

Output:

```
Output string after concatenation: HelloWorld
```

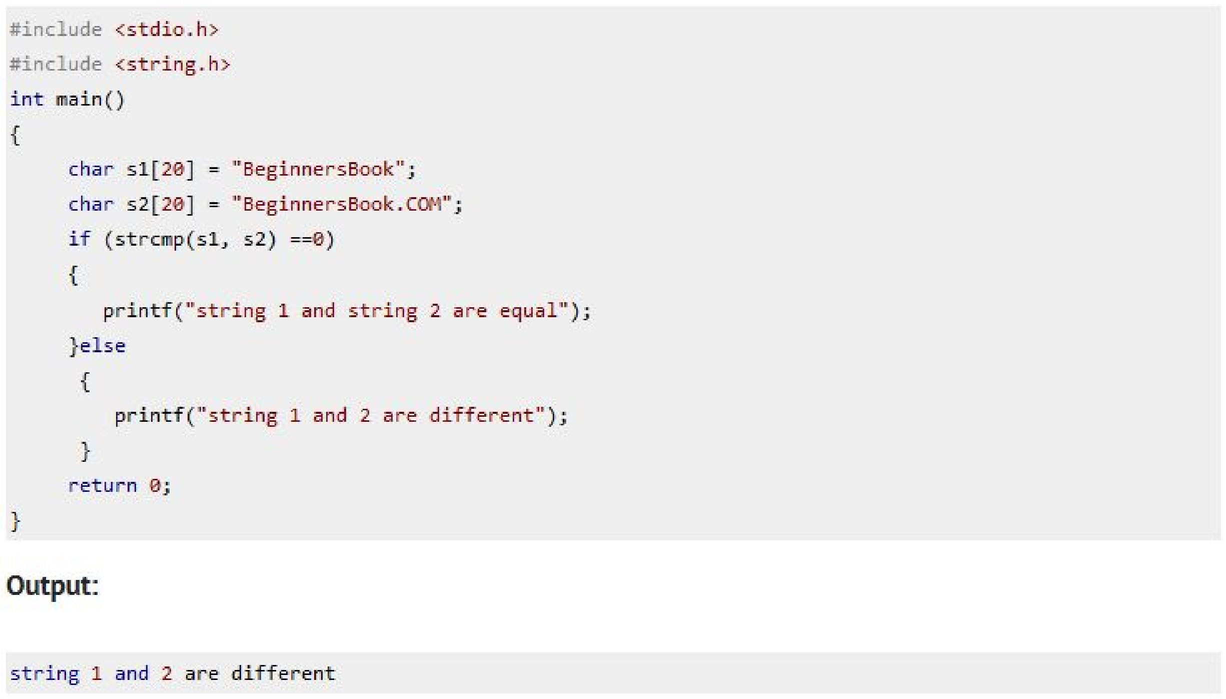
3. Example of strcpy() function

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

int main( )
{
    char source[ ] = "fresh2refresh" ;
    char target[20]= "" ;
    printf ( "\nsource string = %s", source ) ;
    printf ( "\ntarget string = %s", target ) ;
    strcpy ( target, source ) ;
    printf ( "\ntarget string after strcpy( ) = %s", target ) ;
    return 0;
}
```

OUTPUT:

```
source string = fresh2refresh
target string =
target string after strcpy( ) = fresh2refresh
```

4. Example of strcmp() function



```
#include <stdio.h>
#include <string.h>
int main()
{
     char s1[20] = "BeginnersBook";
     char s2[20] = "BeginnersBook.COM";
     if (strcmp(s1, s2) ==0)
     {
        printf("string 1 and string 2 are equal");
     }else
      {
         printf("string 1 and 2 are different");
      }
     return 0;
}
```

Output:

```
string 1 and 2 are different
```

5. Example of strupr() function
6. Example of strlwr() function

Operators:

Operators are symbols or combinations of symbols that directs the computer to perform some operation upon operands. C includes large number of operators, which fall into several different categories.

- Arithmetic Operators
- Assignment Operators
- Unary Operators
- Comparison (Relational) Operators
- Shift Operators

- Bit-wise Operators
- Logical Operators
- Miscellaneous Operators

Arithmetic Operators

There are five arithmetic Operators in C. They are:

Suppose variable a holds 10 and b holds 20

Operator	Description	Expression	Value
+	Adds two operands	a+b	30
-	Subtracts second operand from the first	a-b	-10
*	Multiplies both operands	a*b	200
/	Divides numerator by de-numerator	b/a	2
%	Modulus Operator (remainder after an integer division)	b%a	0

Suppose X1 and X2 are character-type variables that represent the character M and U respectively. Some arithmetic expressions that make use of these variables are shown below.

X1 + X2 =162
X1+ X2+ ‘5’ = 215

Note that M is encoded as 77 , U is encoded as 85, and 5 is encoded as 53 in ASCII character set.

Assignment Operator:

Assignment operator assigns the value of right operator or expression to the variable in left side. There are many variations of assignment operator as described below;

Operator	Description	Example
=	Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Adds the operands and assigns the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*=	Multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A

/=	Divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A

Unary Operators:

A unary operator in C, is an operator that takes a single operand in an expression or a statement. There are basically two unary operators in C. They are:

Operator	Description	Example	Explanation
++	Increases the value of the operand by one.	a++	Equivalent to a=a+1
--	Decreases the value of the operand by one.	a--	Equivalent to a=a-1

The increment operator ++, can be used in two ways:

- As a prefix
  In prefix, the operator precedes the variable i.e. ++var . In this form the value of the variable is first incremented and then used in the expression as illustrated below;

  var1=20;   var2=++var1;

This code is equivalent to the following set of codes:

var1=20;   var1=var1+1;   var2=var1;

At the end, both variables var1 and var2 store value 21.

- As a postfix

  Likewise, in postfix, the operator follows the variable i.e. var++. In this form, the value of variable is used in the expression and then incremented as illustrated below;

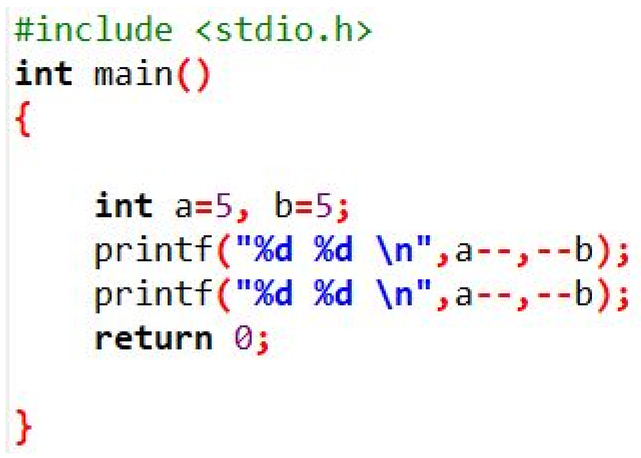
  var1=20;   var2=var1++;

  The equivalent of this code is:

  var1=20;   var2=var1;   var1=var1+1;

  At the end, var1 has the value 21 while var2 reminds set to 20.

Example: Post decrement and Pre decrement

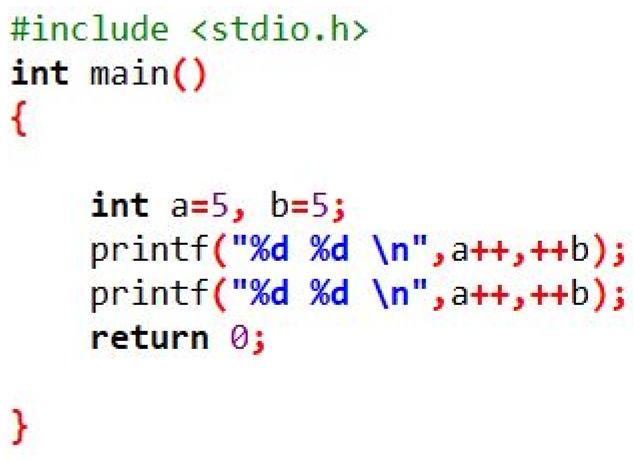


```c
#include <stdio.h>
int main()
{

    int a=5, b=5;
    printf("%d %d \n",a--,--b);
    printf("%d %d \n",a--,--b);
    return 0;

}
```

Output:

```
5 4

4 3
```

Example: Post Increment and Pre Increment



```c
#include <stdio.h>
int main()
{

    int a=5, b=5;
    printf("%d %d \n",a++,++b);
    printf("%d %d \n",a++,++b);
    return 0;

}
```

Output:

```
5 6

6 7
```

Comparison Operator

Comparison operators evaluate to true or false. They are also called relational operators.

Assume variable A holds 10 and variable B holds 20, then

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.

<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Shift Operators

Data is stored internally in binary format. A bit can have a value of one or zero. Eight bits form a byte. Shift operators work on individual bits in a byte. Using the shift operator involves moving the bit pattern left or right. We can use them only on integer data type and not on the char , float , or double data types.

Operator	Description	Example	Explanation
>>	Shifts bits to the right, filling sign bir at the left.	a=10>>3	The result of this is 10 divided by 2^3.
<<	Shifts bits to the	a=10<<3	The result if this is

	left, filling zeros at the right.		10 multiplied by 2^3.

If the int data type occupies four byte in the memory, the rightmost eight bits of the number are represented in binary as “00001010”

When we do right shift by 3, the result is “00000001” which is equivalent to 1 and when we do left shift by 3, the result is “01010000” which is equivalent to 80.

Bit-wise operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; now in binary format they will be as follows −

A = 0011 1100

B = 0000 1101

Binary And Operator:

A&B = 0000 1100

Binary Or Operator:

A|B = 0011 1101

Binary XOR Operator:

A^B = 0011 0001

Binary Ones Complement Operator:

~A = 1100 0011

Logical Operators

There are following logical operators supported by C language.

Assume variable A holds 1 and variable B holds 0, then

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.

||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

Miscellaneous Operators

The following table lists some other operators that C supports.

Sr.No	Operator & Description
1	sizeof **sizeof operator** returns the size of a variable. For example, sizeof(a), where 'a' is integer, and will return 4.
2	Condition ? X : Y **Conditional operator (?)**. If Condition is true then it returns value of X otherwise returns value of Y. Example:

	result: (marks>50 ? “pass”: “fail”) ; In above example, if the marks obtained is more than 50, result will be pass else fail.
3	, **Comma operator** causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list.
4	. (dot) and -> (arrow) **Member operators** are used to reference individual members of classes, structures, and unions.
5	Cast **Casting operators** convert one data type to another. For example, int(2.2000) would return 2.
6	& **Pointer operator &** returns the address of a variable. For example &a; will give actual address of the variable.

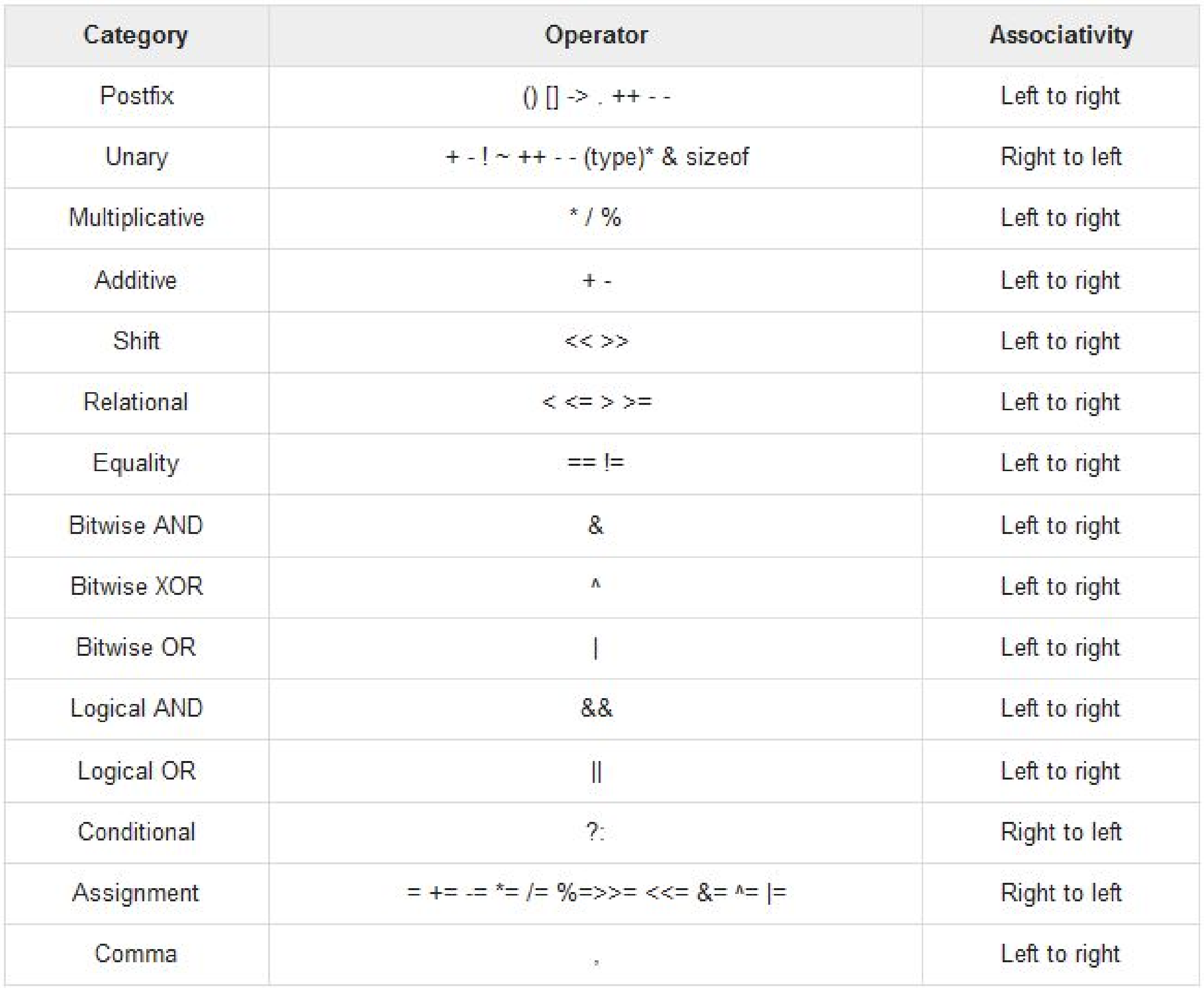
7	* **Pointer operator *** is pointer to a variable. For example *var; will pointer to a variable var.

Precedence of Operators

If more than one operators are involved in an expression, C language has a predefined rule of priority for the operators. This rule of priority of operators is called operator precedence.

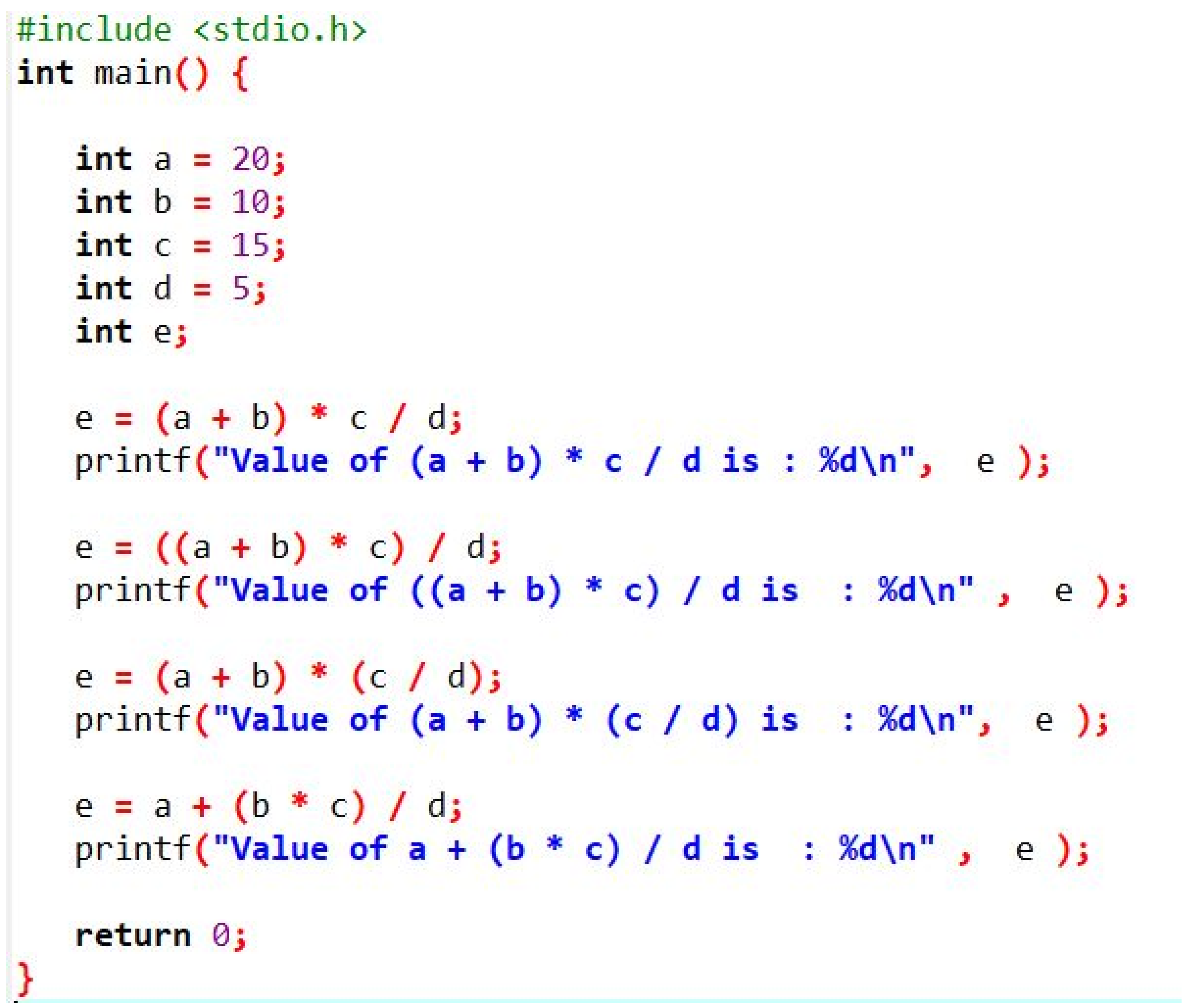
In C, precedence of arithmetic operators(*, %, /, +, -) is higher than relational operators(==, !=, >, <, >=, <=) and precedence of relational operator is higher than logical operators(&&, || and !).

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.



Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Example:



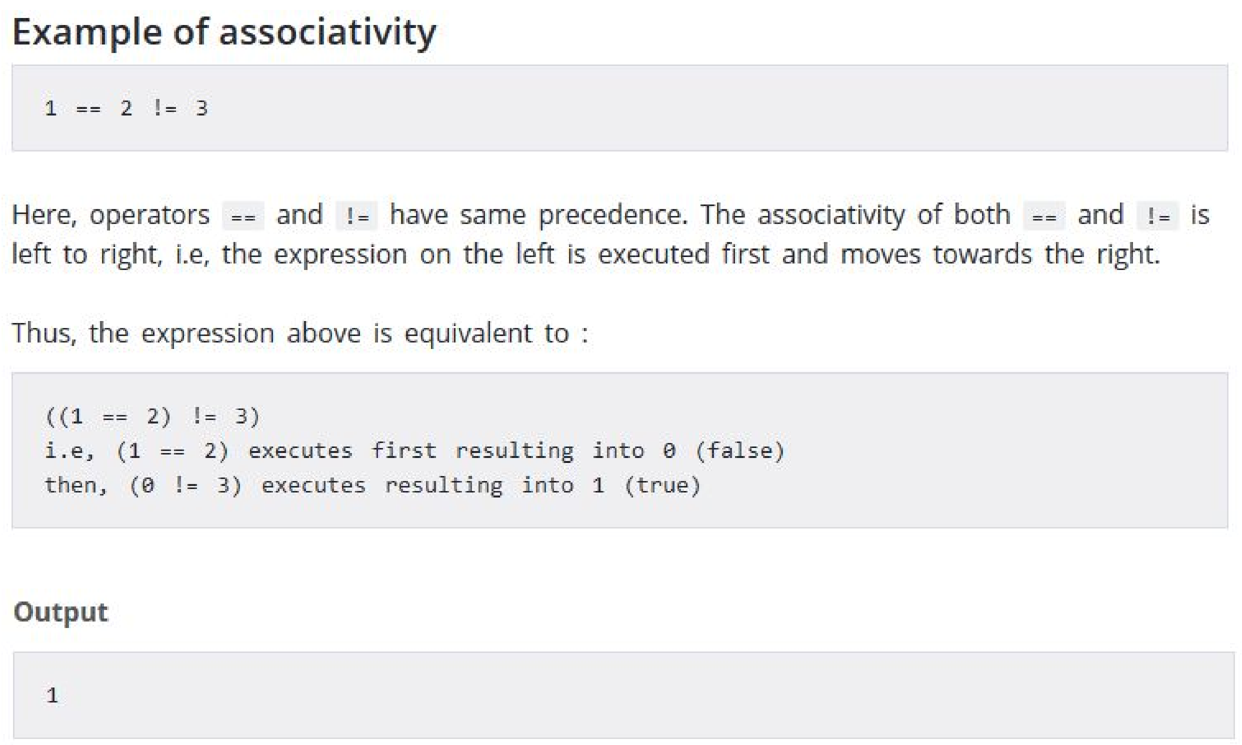
```c
#include <stdio.h>
int main() {

   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;

   e = (a + b) * c / d;
   printf("Value of (a + b) * c / d is : %d\n",  e );

   e = ((a + b) * c) / d;
   printf("Value of ((a + b) * c) / d is  : %d\n" ,  e );

   e = (a + b) * (c / d);
   printf("Value of (a + b) * (c / d) is  : %d\n",  e );

   e = a + (b * c) / d;
   printf("Value of a + (b * c) / d is  : %d\n" ,  e );

   return 0;
}
```

Output:

```
Value of (a + b) * c / d is : 90

Value of ((a + b) * c) / d is  : 90

Value of (a + b) * (c / d) is  : 90

Value of a + (b * c) / d is  : 50
```

Associativity of Operators

If two operators of same precedence (priority) is present in an expression, Associativity of operators indicate the order in which they execute.



Example of associativity

```
1 == 2 != 3
```

Here, operators `==` and `!=` have same precedence. The associativity of both `==` and `!=` is left to right, i.e, the expression on the left is executed first and moves towards the right.

Thus, the expression above is equivalent to :

```
((1 == 2) != 3)
i.e, (1 == 2) executes first resulting into 0 (false)
then, (0 != 3) executes resulting into 1 (true)
```

Output

```
1
```

Format Specifier in C

The format specifier is used during input and output. It is a way to tell the compiler what type of data is in a variable during taking input using scanf() or printing using printf().

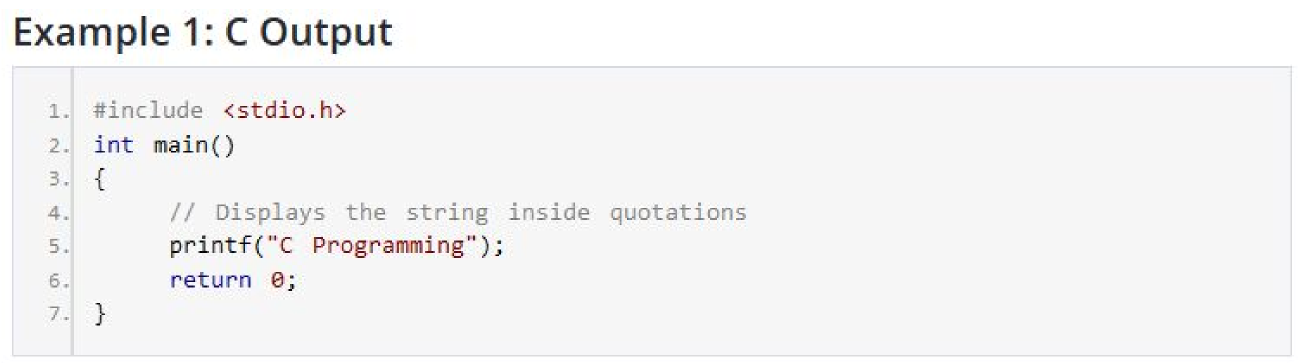
List of Format Specifiers in C:

Data Type	Format Specifier
int	%d
char	%c

float	%f
double	%lf
long int	%li
long long int	%lli
string	%s

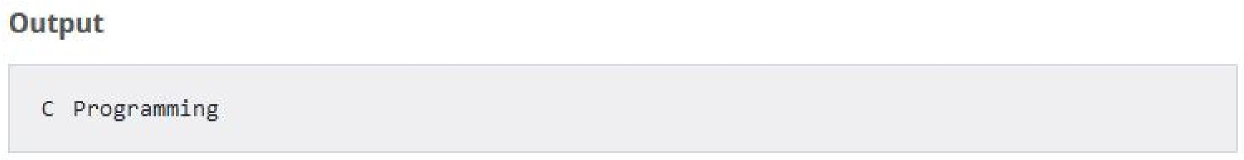
C output using printf()

In C programming, printf() is one of the main output function. The function sends formatted output to the screen. For example,



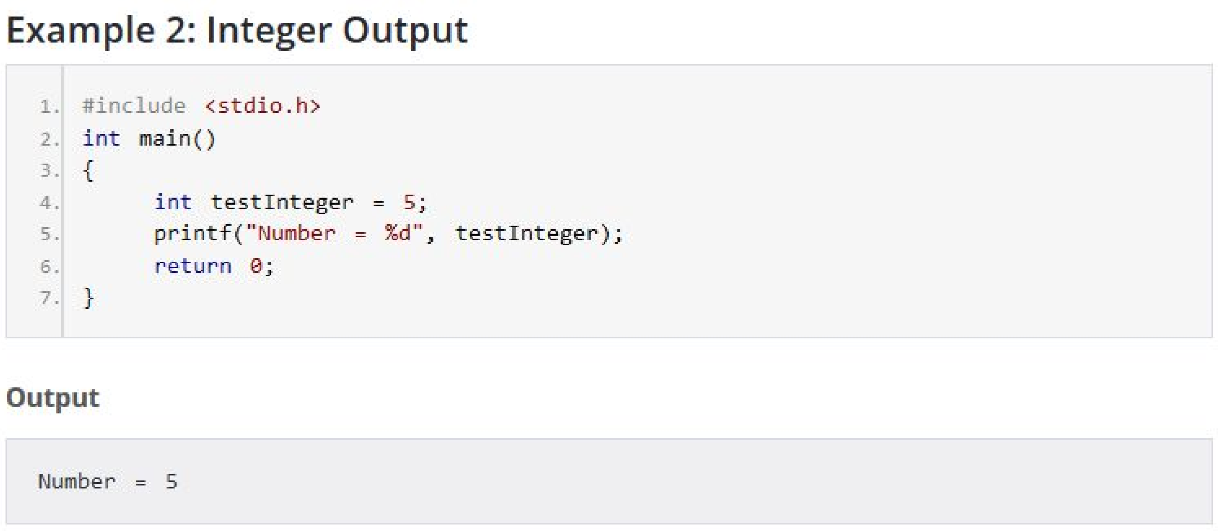
Example 1: C Output

```c
#include <stdio.h>
int main()
{
      // Displays the string inside quotations
      printf("C Programming");
      return 0;
}
```



Output

```
C Programming
```



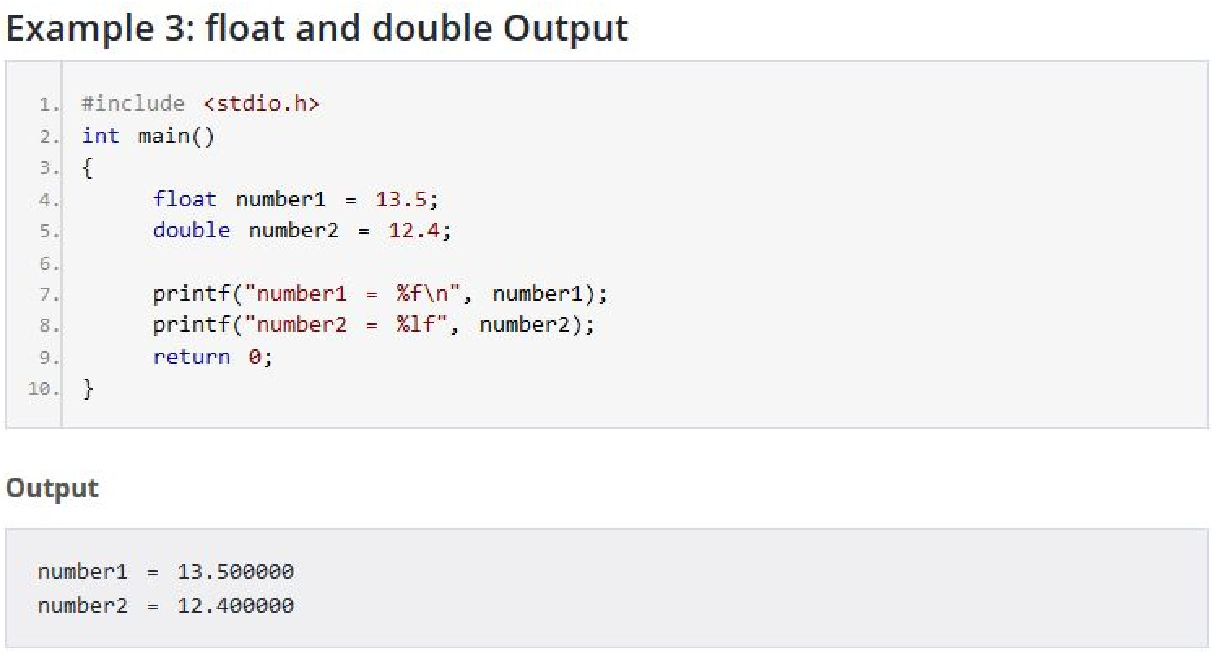
Example 2: Integer Output

```c
#include <stdio.h>
int main()
{
      int testInteger = 5;
      printf("Number = %d", testInteger);
      return 0;
}
```

Output

```
Number = 5
```

We use %d format specifier to print int types. Here, the %d inside the quotations will be replaced by the value of testInteger.



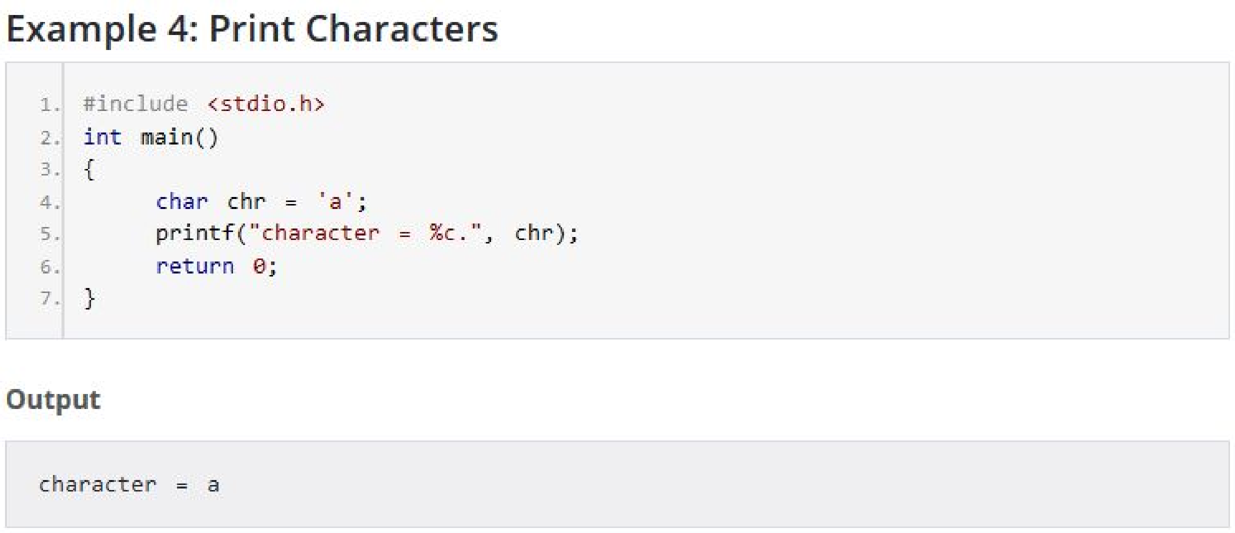
Example 3: float and double Output

```c
#include <stdio.h>
int main()
{
      float number1 = 13.5;
      double number2 = 12.4;

      printf("number1 = %f\n", number1);
      printf("number2 = %lf", number2);
      return 0;
}
```

Output

```
number1 = 13.500000
number2 = 12.400000
```

To print float, we use %f format specifier. Similarly, we use %lf to print double values.



Example 4: Print Characters

```c
#include <stdio.h>
int main()
{
      char chr = 'a';
      printf("character = %c.", chr);
      return 0;
}
```

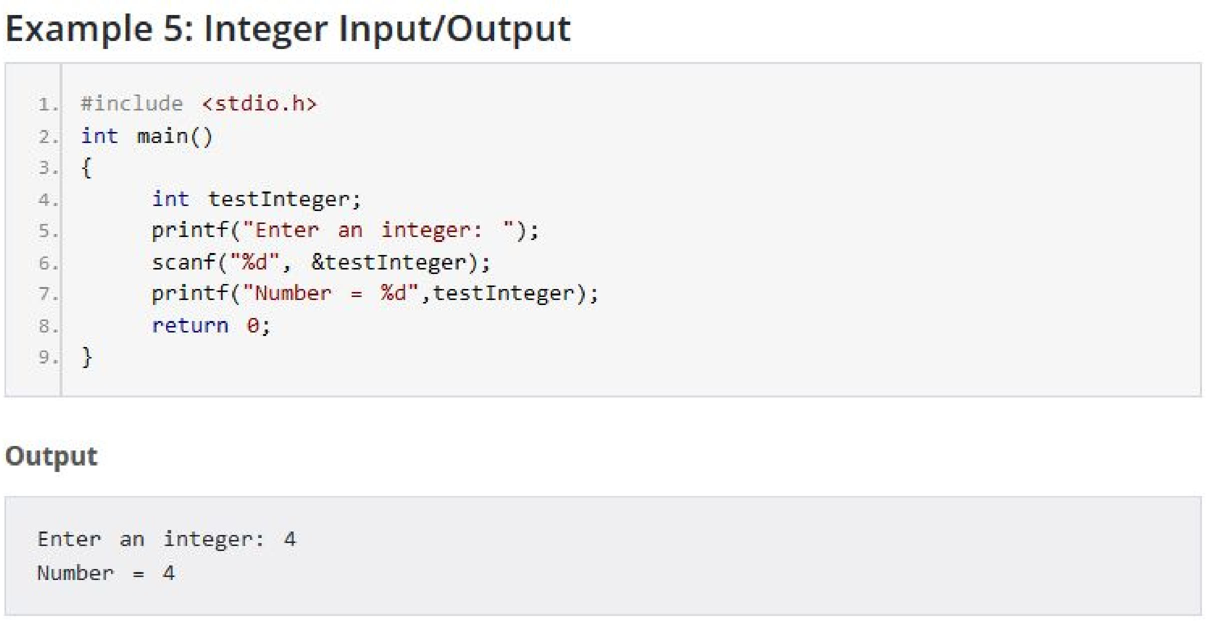
Output

```
character = a
```

To print char, we use %c format specifier.

C Input using scanf()

In C programming, scanf() is one of the commonly used function to take input from the user. The scanf() function reads formatted input from the standard input such as keyboards.



Example 5: Integer Input/Output

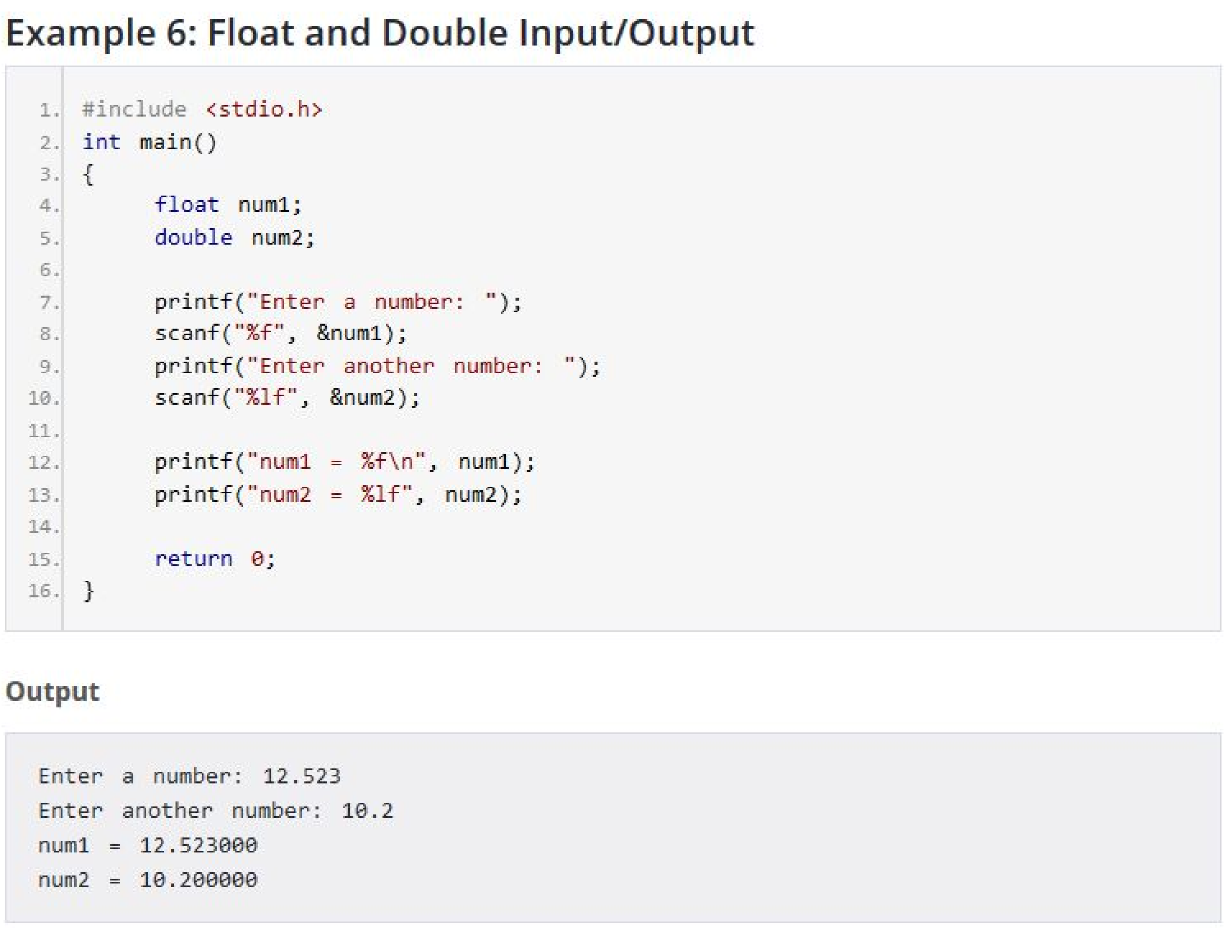
```c
#include <stdio.h>
int main()
{
      int testInteger;
      printf("Enter an integer: ");
      scanf("%d", &testInteger);
      printf("Number = %d",testInteger);
      return 0;
}
```

Output

```
Enter an integer: 4
Number = 4
```

Here, we have used %d format specifier inside the scanf() function to take int input from the user. When the user enters an integer, it is stored in the testInteger variable.

Notice, that we have used &testInteger inside scanf(). It is because &testInteger gets the address of testInteger, and the value entered by the user is stored in that address.



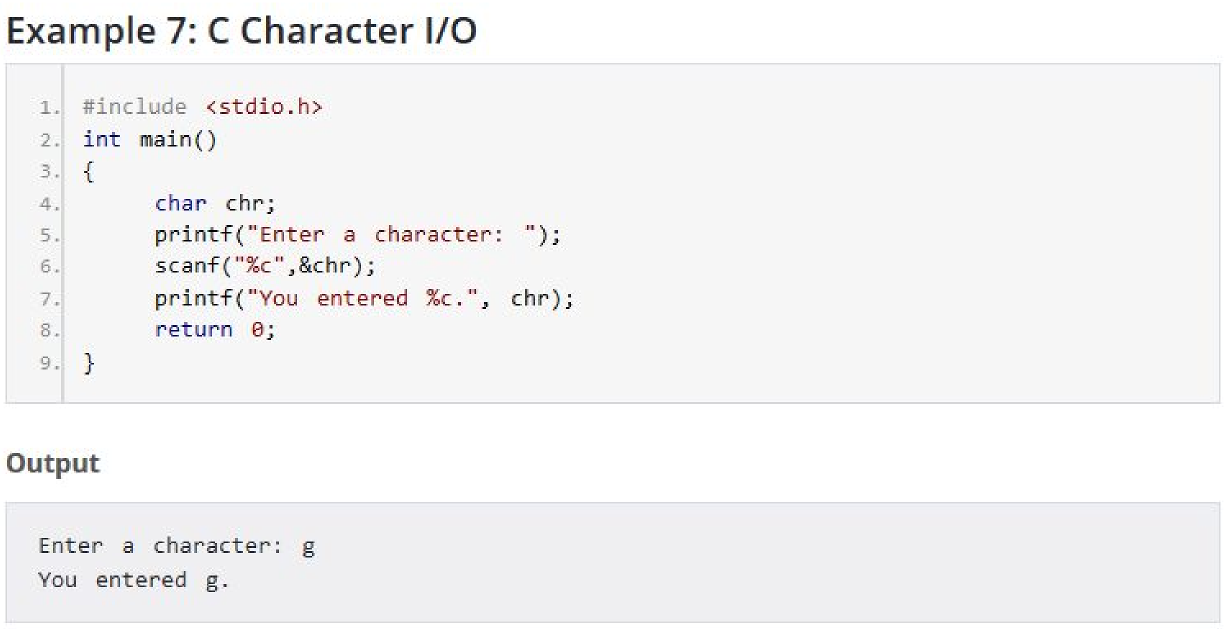
Example 6: Float and Double Input/Output

```c
#include <stdio.h>
int main()
{
     float num1;
     double num2;

     printf("Enter a number: ");
     scanf("%f", &num1);
     printf("Enter another number: ");
     scanf("%lf", &num2);

     printf("num1 = %f\n", num1);
     printf("num2 = %lf", num2);

     return 0;
}
```

Output

```
Enter a number: 12.523
Enter another number: 10.2
num1 = 12.523000
num2 = 10.200000
```

We use %f and %lf format specifier for float and double respectively.



Example 7: C Character I/O

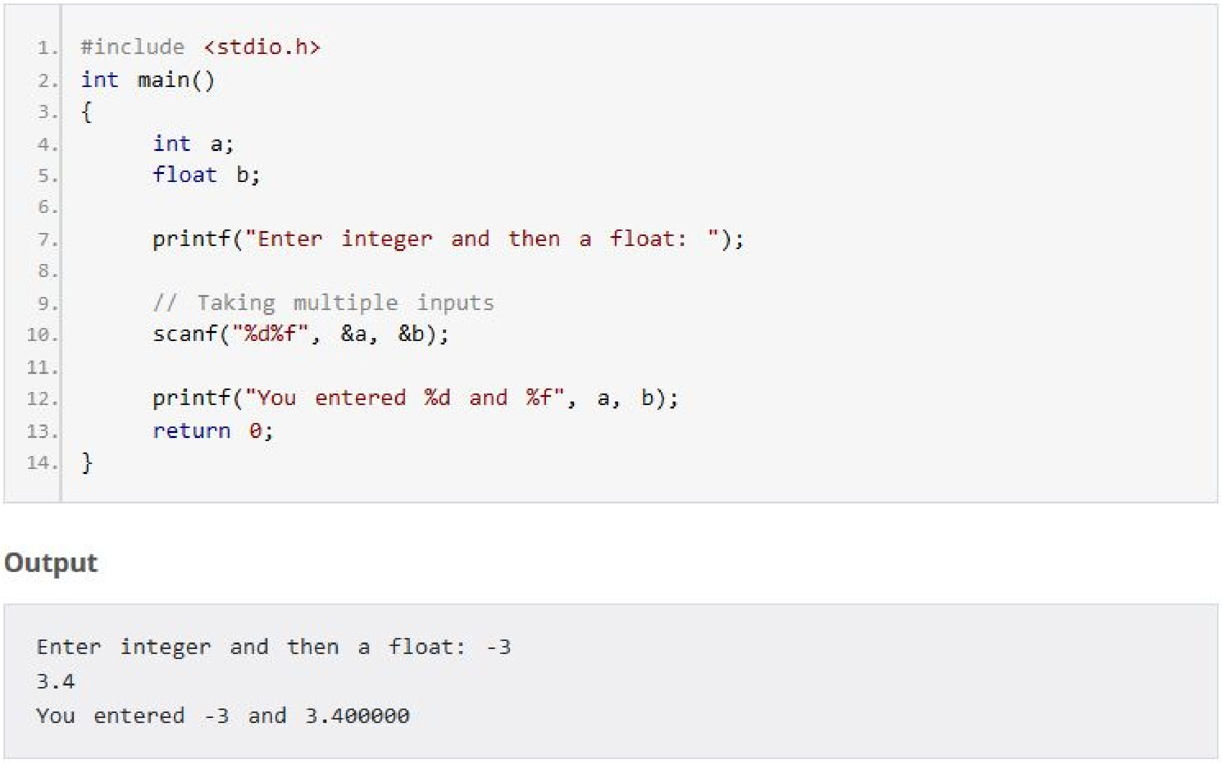
```c
#include <stdio.h>
int main()
{
     char chr;
     printf("Enter a character: ");
     scanf("%c",&chr);
     printf("You entered %c.", chr);
     return 0;
}
```

Output

```
Enter a character: g
You entered g.
```

I/O Multiple Values

Here's how you can take multiple inputs from the user and display them.



```c
#include <stdio.h>
int main()
{
     int a;
     float b;

     printf("Enter integer and then a float: ");

     // Taking multiple inputs
     scanf("%d%f", &a, &b);

     printf("You entered %d and %f", a, b);
     return 0;
}
```

Output

```
Enter integer and then a float: -3
3.4
You entered -3 and 3.400000
```

Escape Characters (Escape Sequence)

An escape sequence is a sequence of characters that does not represent itself ,but is translated into a sequence of characters that may be difficult or impossible to represent directly. Here is the list of such escape codes:

Escape Sequence	Meaning	Elucidation
\n	New Line	Used to shift the cursor control to the new line
\t	Horizontal Tab	Used to shift the cursor to a couple of spaces to the right in the same line.
\a	Audible bell	A beep is generated indicating the execution of the program to alert the user.
\r	Carriage return	Used to position the cursor to the beginning of the current line.
\\	Backslash	Used to display the backslash character
\?	Question Mark	Used to print the question mark
\’	Single Quote	Used to print single quote
\”	Double Quote	Used to print double quote
\0	Null Character	Used for Null Characters

https://www.w3schools.in/c-tutorial/operators/