C Operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise, etc.

There are following types of operators to perform different types of operations in C language.

- Arithmetic Operators
- Relational Operators
- Shift Operators
- Logical Operators
- Bitwise Operators
- Ternary or Conditional Operators
- Assignment Operator
- Misc Operator

Precedence of Operators in C

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operator direction to be evaluated; it may be left to right or right to left.

Let's understand the precedence by the example given below:

int value=10+20*10;

The value variable will contain **210** because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C operators is given below:

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

Comments in C

Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in the C language.

1. Single Line Comments
2. Multi-Line Comments

Single Line Comments

Single line comments are represented by double slash \\. Let's see an example of a single line comment in C.

```
#include<stdio.h>
int main(){
   //printing information
   printf("Hello C");
return 0;
}
```

Output:

```
Hello C
```

Even you can place the comment after the statement. For example:

printf("Hello C");//printing information

Mult Line Comments

Multi-Line comments are represented by slash asterisk * ... *\. It can occupy many lines of code, but it can't be nested. Syntax:

```
/*
code
to be commented
*/
```

Let's see an example of a multi-Line comment in C.

```
#include<stdio.h>
int main(){
    /*printing information
      Multi-Line Comment*/
    printf("Hello C");
return 0;
}
```

Output:

```
Hello C
```

C Format Specifier

The Format specifier is a string used in the formatted input and output functions. The format string determines the format of the input and output. The format string always starts with a '%' character.

The commonly used format specifiers in printf() function are:

Format specifier	**Description**
%d or %i	It is used to print the signed integer value where signed integer means that the variable can hold both positive and negative values.
%u	It is used to print the unsigned integer value where the unsigned integer means that the variable can hold only positive value.
%o	It is used to print the octal unsigned integer where octal integer value always starts with a 0 value.
%x	It is used to print the hexadecimal unsigned integer where the hexadecimal integer value always starts with a 0x value. In this, alphabetical characters are printed in small letters such as a, b, c, etc.
%X	It is used to print the hexadecimal unsigned integer, but %X prints the alphabetical characters in uppercase such as A, B, C, etc.
%f	It is used for printing the decimal floating-point values. By default, it prints the 6 values after '.'.
%e/%E	It is used for scientific notation. It is also known as Mantissa or Exponent.
%g	It is used to print the decimal floating-point values, and it uses the fixed precision, i.e., the value after the decimal in input would be exactly the same as the value in the output.
%p	It is used to print the address in a hexadecimal form.
%c	It is used to print the unsigned character.
%s	It is used to print the strings.
%ld	It is used to print the long-signed integer value.

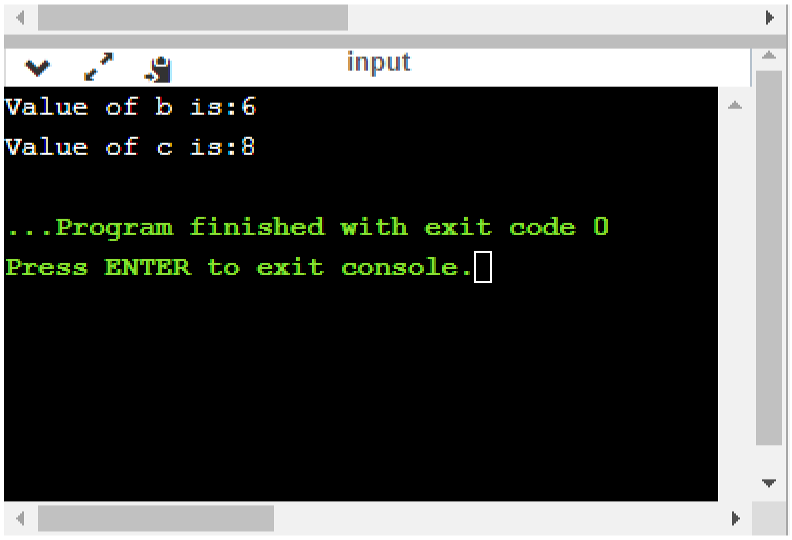
Let's understand the format specifiers in detail through an example.

- **%d**

```
int main()
{
 int b=6;
 int c=8;
 printf("Value of b is:%d", b);
 printf("\nValue of c is:%d",c);
    return 0;
}
```

In the above code, we are printing the integer value of b and c by using the %d specifier.

Output



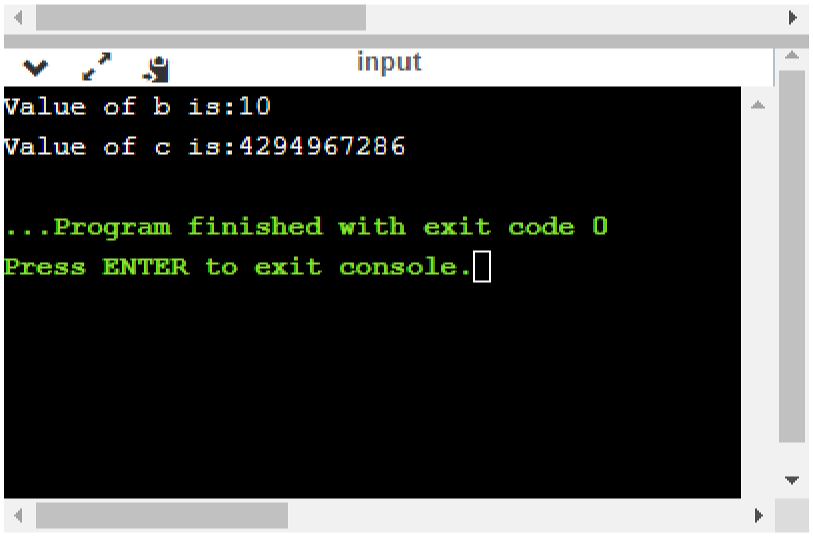
```
Value of b is:6
Value of c is:8

...Program finished with exit code 0
Press ENTER to exit console.
```

- **%u**

```
int main()
{
 int b=10;
 int c= -10;
 printf("Value of b is:%u", b);
 printf("\nValue of c is:%u",c);
    return 0;
}
```

In the above program, we are displaying the value of b and c by using an unsigned format specifier, i.e., %u. The value of b is positive, so %u specifier prints the exact value of b, but it does not print the value of c as c contains the negative value.

Output



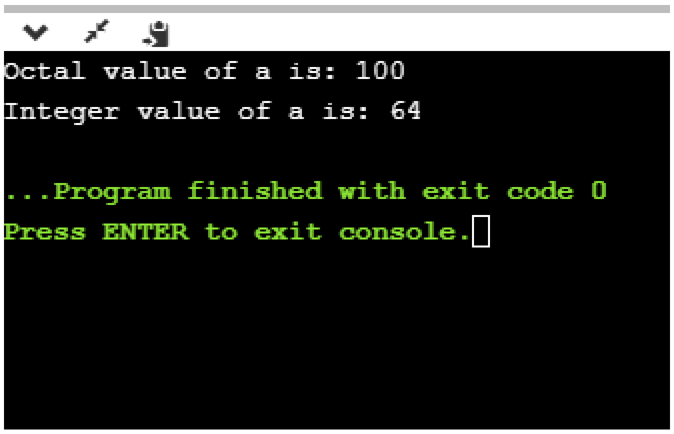
```
Value of b is:10
Value of c is:4294967286

...Program finished with exit code 0
Press ENTER to exit console.
```

- **%o**

```
int main()
{
  int a=0100;
  printf("Octal value of a is: %o", a);
  printf("\nInteger value of a is: %d",a);
  return 0;
}
```

In the above code, we are displaying the octal value and integer value of a.

Output



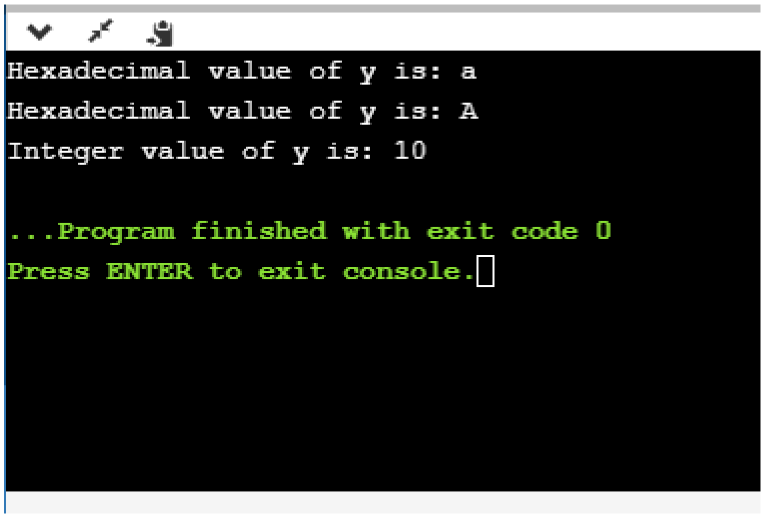
```
Octal value of a is: 100
Integer value of a is: 64

...Program finished with exit code 0
Press ENTER to exit console.
```

- **%x and %X**

```
int main()
{
  int y=0xA;
  printf("Hexadecimal value of y is: %x", y);
  printf("\nHexadecimal value of y is: %X",y);
  printf("\nInteger value of y is: %d",y);
    return 0;
}
```

In the above code, y contains the hexadecimal value 'A'. We display the hexadecimal value of y in two formats. We use %x and %X to print the hexadecimal value where %x displays the value in small letters, i.e., 'a' and %X displays the value in a capital letter, i.e., 'A'.

Output



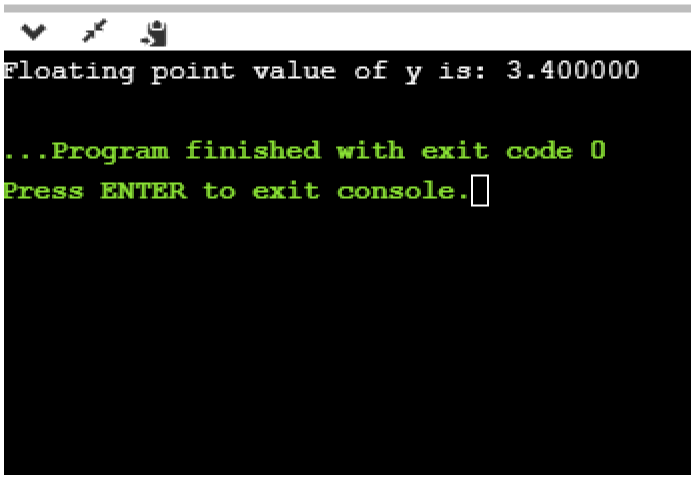
```
Hexadecimal value of y is: a
Hexadecimal value of y is: A
Integer value of y is: 10

...Program finished with exit code 0
Press ENTER to exit console.
```

- **%f**

```
int main()
{
  float y=3.4;
  printf("Floating point value of y is: %f", y);
  return 0;
}
```

The above code prints the floating value of y.

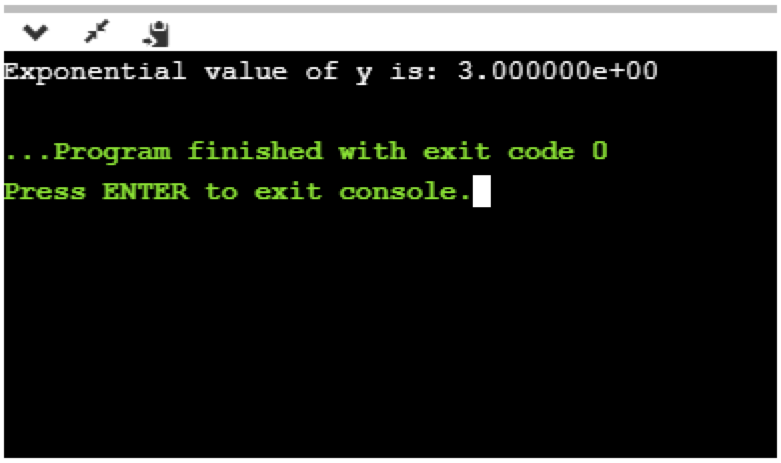
Output


```
Floating point value of y is: 3.400000

...Program finished with exit code 0
Press ENTER to exit console.
```

- **%e**

```
int main()
{
  float y=3;
  printf("Exponential value of y is: %e", y);
  return 0;
}
```

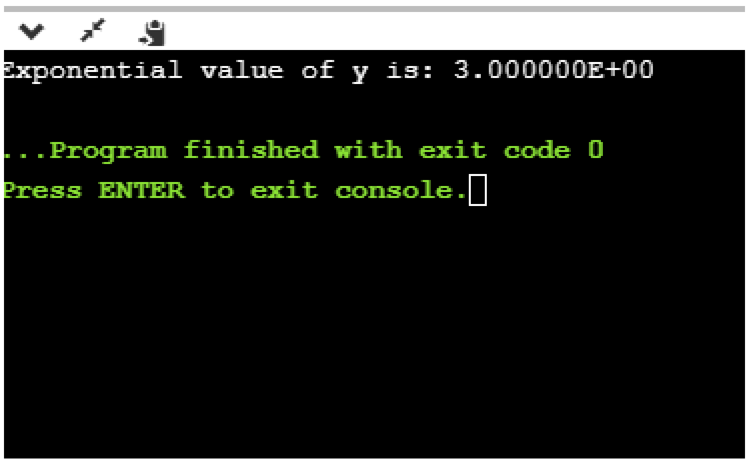
Output


```
Exponential value of y is: 3.000000e+00

...Program finished with exit code 0
Press ENTER to exit console.
```

- **%E**

```
int main()
{
  float y=3;
  printf("Exponential value of y is: %E", y);
  return 0;
}
```

Output



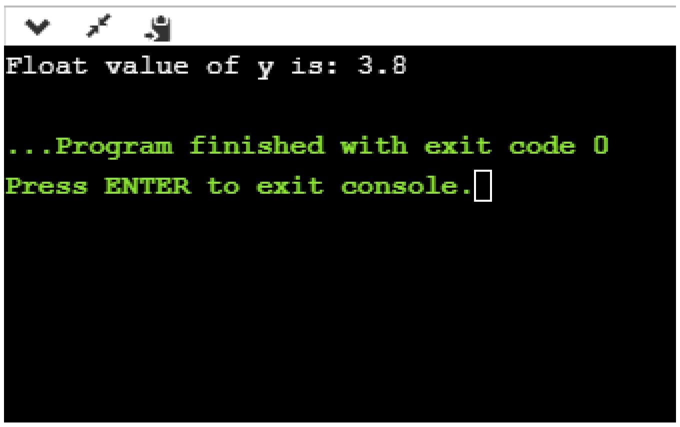
```
Exponential value of y is: 3.000000E+00

...Program finished with exit code 0
Press ENTER to exit console.
```

- **%g**

```
int main()
{
  float y=3.8;
  printf("Float value of y is: %g", y);
  return 0;
}
```

In the above code, we are displaying the floating value of y by using %g specifier. The %g specifier displays the output same as the input with a same precision.

Output

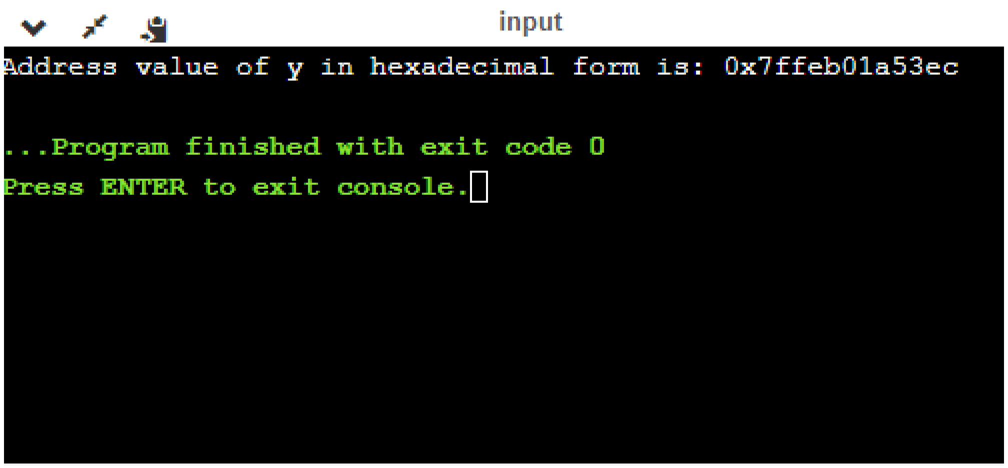


```
Float value of y is: 3.8

...Program finished with exit code 0
Press ENTER to exit console.
```

- **%p**

```
int main()
{
  int y=5;
  printf("Address value of y in hexadecimal form is: %p", &y);
  return 0;
}
```

Output

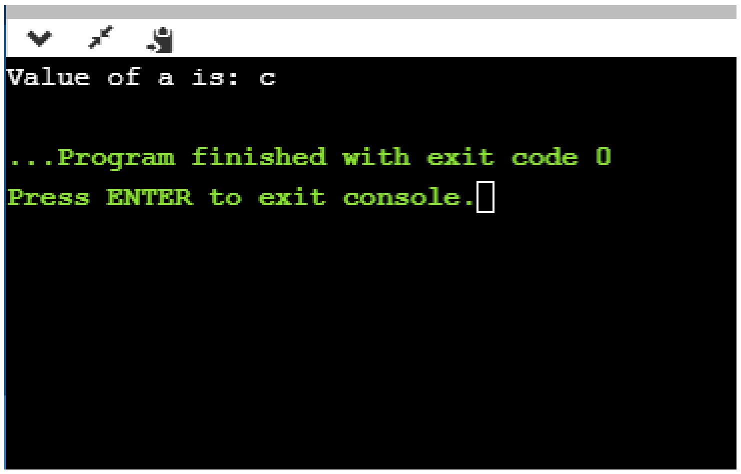


```
Address value of y in hexadecimal form is: 0x7ffeb01a53ec

...Program finished with exit code 0
Press ENTER to exit console.
```

- **%c**

```
int main()
{
  char a='c';
  printf("Value of a is: %c", a);
  return 0;
}
```

Output



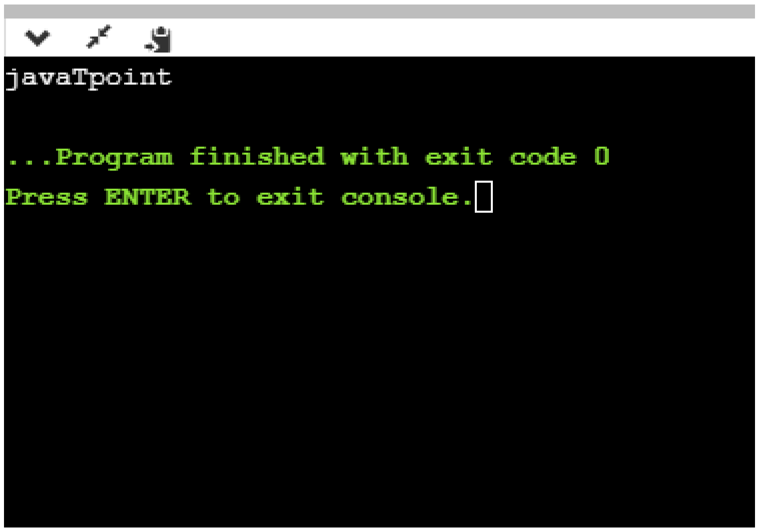
```
Value of a is: c

...Program finished with exit code 0
Press ENTER to exit console.
```

- **%s**

```
int main()
```

```
{
  printf("%s", "javaTpoint");
  return 0;
}
```

Output



```
javaTpoint

...Program finished with exit code 0
Press ENTER to exit console.
```

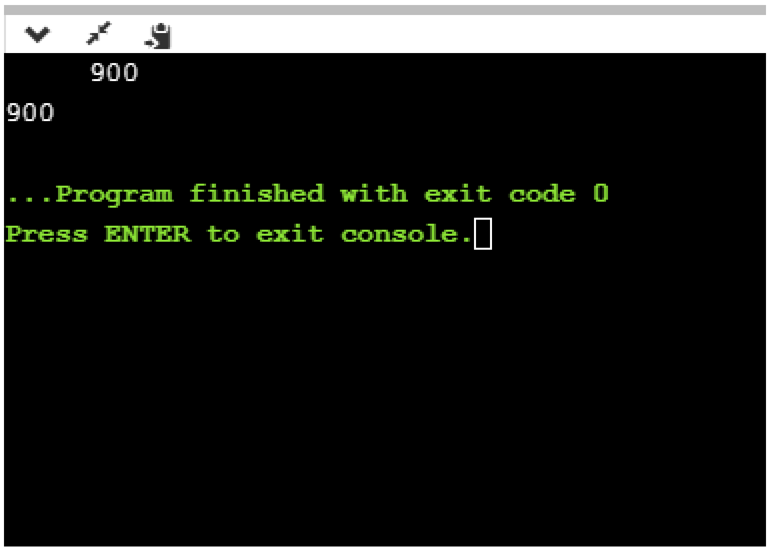
Minimum Field Width Specifier

Suppose we want to display an output that occupies a minimum number of spaces on the screen. You can achieve this by displaying an integer number after the percent sign of the format specifier.

```
int main()
{
 int x=900;
  printf("%8d", x);
  printf("\n%-8d",x);
  return 0;
}
```

In the above program, %8d specifier displays the value after 8 spaces while %-8d specifier will make a value left-aligned.

Output



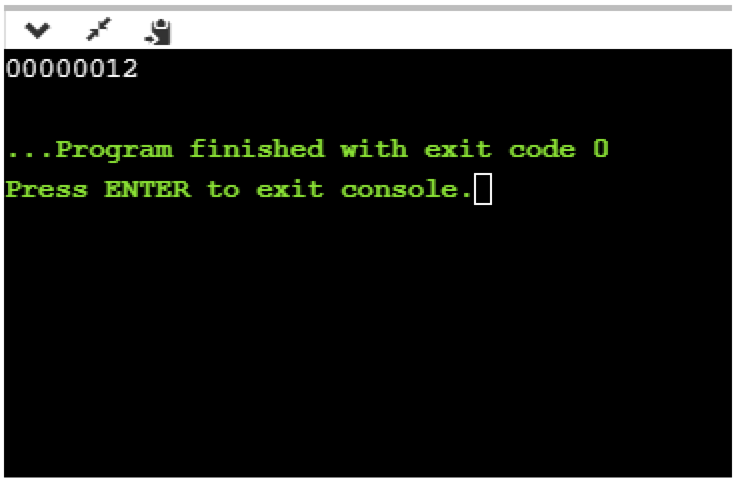
```
     900
900

...Program finished with exit code 0
Press ENTER to exit console.
```

Now we will see how to fill the empty spaces. It is shown in the below code:

```
int main()
{
 int x=12;
  printf("%08d", x);
  return 0;
}
```

In the above program, %08d means that the empty space is filled with zeroes.

Output



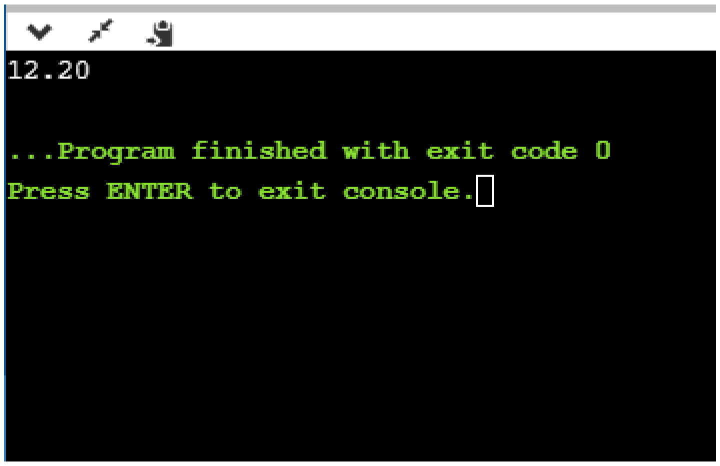
```
00000012

...Program finished with exit code 0
Press ENTER to exit console.
```

Specifying Precision

We can specify the precision by using '.' (Dot) operator which is followed by integer and format specifier.

```
int main()
{
 float x=12.2;
  printf("%.2f", x);
  return 0;
}
```

Output



```
12.20

...Program finished with exit code 0
Press ENTER to exit console.
```

Escape Sequence in C

An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character.

It is composed of two or more characters starting with backslash \. For example: \n represents new line.

List of Escape Sequences in C

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)

\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

Escape Sequence Example

```
#include<stdio.h>
int main(){
    int number=50;
   printf("You\nare\nlearning\n\'c\' language\n\"Do you know C language\"");
return 0;
}
```

Output:

```
You
are
learning
'c' language
"Do you know C language"
```