Topic:- Comments

C Format Specifier

Escape Sequence in C

ASCII value in C

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.

1. #include<stdio.h>
2. **int** main(){
3. //printing information
4. printf("Hello C");
5. **return** 0;
6. }

Output:

Hello C

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

1. 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:

1. /\*
2. code
3. to be commented
4. \*/

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

1. #include<stdio.h>
2. **int** main(){
3. /\*printing information
4. Multi-Line Comment\*/
5. printf("Hello C");
6. **return** 0;
7. }

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**

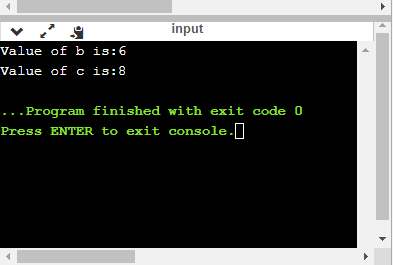
1. **int** main()
2. {
3. **int** b=6;
4. **int** c=8;
5. printf("Value of b is:%d", b);
6. printf("\nValue of c is:%d",c);
8. **return** 0;
9. }

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

**Output**

52.1M

953

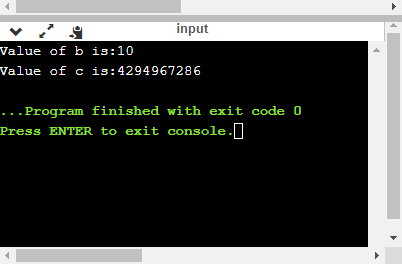
Java Try Catch

* **%u**

1. **int** main()
2. {
3. **int** b=10;
4. **int** c= -10;
5. printf("Value of b is:%u", b);
6. printf("\nValue of c is:%u",c);
8. **return** 0;
9. }

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**

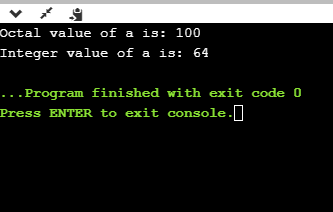


* **%o**

1. **int** main()
2. {
3. **int** a=0100;
4. printf("Octal value of a is: %o", a);
5. printf("\nInteger value of a is: %d",a);
6. **return** 0;
7. }

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

**Output**

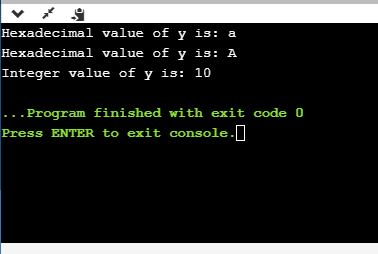


* **%x and %X**

1. **int** main()
2. {
3. **int** y=0xA;
4. printf("Hexadecimal value of y is: %x", y);
5. printf("\nHexadecimal value of y is: %X",y);
6. printf("\nInteger value of y is: %d",y);
7. **return** 0;
8. }

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**

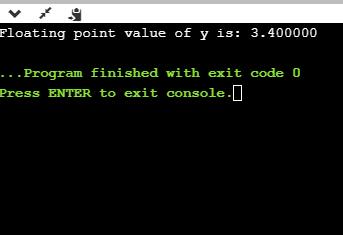


* **%f**

1. **int** main()
2. {
3. **float** y=3.4;
4. printf("Floating point value of y is: %f", y);
5. **return** 0;
6. }

The above code prints the floating value of y.

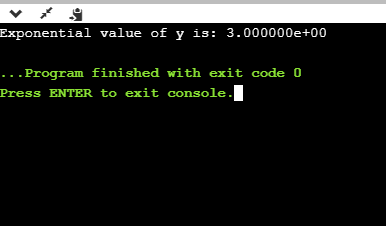
**Output**



* **%e**

1. **int** main()
2. {
3. **float** y=3;
4. printf("Exponential value of y is: %e", y);
5. **return** 0;
6. }

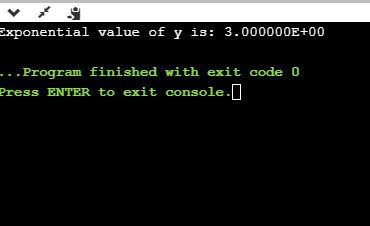
**Output**



* **%E**

1. **int** main()
2. {
3. **float** y=3;
4. printf("Exponential value of y is: %E", y);
5. **return** 0;
6. }

**Output**

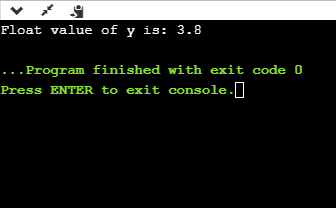


* **%g**

1. **int** main()
2. {
3. **float** y=3.8;
4. printf("Float value of y is: %g", y);
5. **return** 0;
6. }

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.

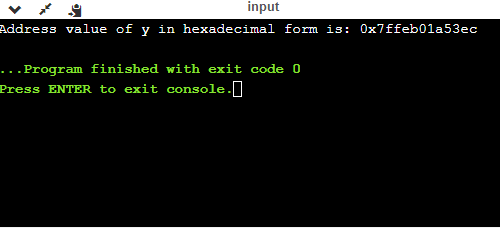
**Output**



* **%p**

1. **int** main()
2. {
3. **int** y=5;
4. printf("Address value of y in hexadecimal form is: %p", &y);
5. **return** 0;
6. }

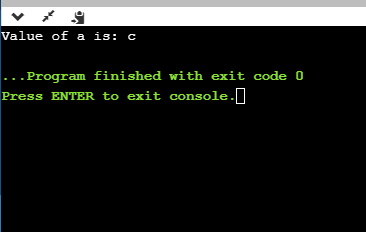
**Output**



* **%c**

1. **int** main()
2. {
3. **char** a='c';
4. printf("Value of a is: %c", a);
5. **return** 0;
6. }

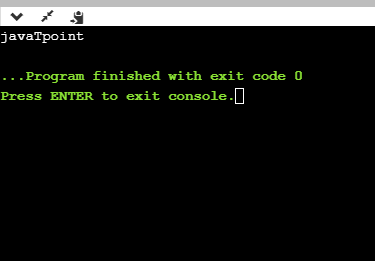
**Output**



* **%s**

1. **int** main()
2. {
3. printf("%s", "javaTpoint");
4. **return** 0;
5. }

**Output**



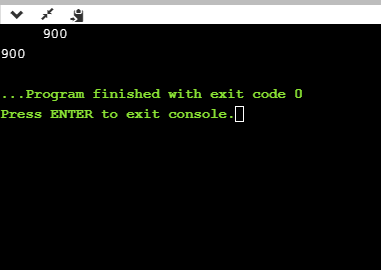
Minimum Field Width Specifier

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.

1. **int** main()
2. {
3. **int** x=900;
4. printf("%8d", x);
5. printf("\n%-8d",x);
6. **return** 0;
7. }

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

**Output**

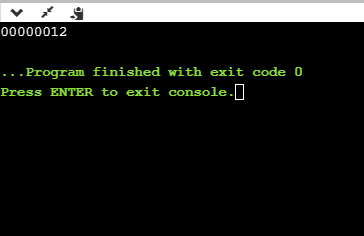


**Now we will see how to fill the empty spaces. It is shown in the below code:**

1. **int** main()
2. {
3. **int** x=12;
4. printf("%08d", x);
5. **return** 0;
6. }

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

**Output**

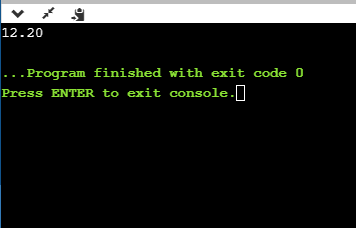


Specifying Precision

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

1. **int** main()
2. {
3. **float** x=12.2;
4. printf("%.2f", x);
5. **return** 0;
6. }

**Output**



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 |
| \0 | Null |

Escape Sequence Example

1. #include<stdio.h>
2. **int** main(){
3. **int** number=50;
4. printf("You\nare\nlearning\n\'c\' language\n\"Do you know C language\"");
5. **return** 0;
6. }

**Output:**

You

are

learning

'c' language

"Do you know C language"

ASCII value in C

What is ASCII code?

The full form of ASCII is the **American Standard Code for information interchange**. It is a character encoding scheme used for electronics communication. Each character or a special character is represented by some ASCII code, and each ascii code occupies 7 bits in memory.

In [C programming language](https://www.javatpoint.com/c-programming-language-tutorial), a character variable does not contain a character value itself rather the ascii value of the character variable. The ascii value represents the character variable in numbers, and each character variable is assigned with some number range from 0 to 127. For example, the ascii value of 'A' is 65.

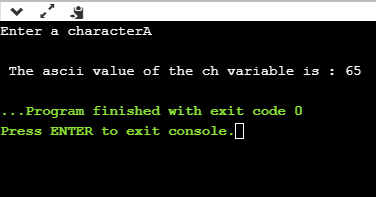
In the above example, we assign 'A' to the character variable whose ascii value is 65, so 65 will be stored in the character variable rather than 'A'.

**We will create a**[**program**](https://www.javatpoint.com/c-programs)**which will display the ascii value of the character variable.**

1. #include <stdio.h>
2. **int** main()
3. {
4. **char** ch;    // variable declaration
5. printf("Enter a character");
6. scanf("%c",&ch);  // user input
7. printf("\n The ascii value of the ch variable is : %d", ch);
8. **return** 0;
9. }

In the above code, the first user will give the character input, and the input will get stored in the 'ch' variable. If we print the value of the 'ch' variable by using %c format specifier, then it will display 'A' because we have given the character input as 'A', and if we use the %d format specifier then its ascii value will be displayed, i.e., 65.

**Output**



The above output shows that the user gave the input as 'A', and after giving input, the ascii value of 'A' will get printed, i.e., 65.

Now, we will create a program which will display the ascii value of all the characters.

1. #include <stdio.h>
2. **int** main()
3. {
4. **int** k;   // variable declaration
5. **for**(**int** k=0;k<=255;k++)  // for loop from 0-255
6. {
7. printf("\nThe ascii value of %c is %d", k,k);
8. }
9. **return** 0;
10. }

The above program will display the ascii value of all the characters. As we know that ascii value of all the characters starts from 0 and ends at 255, so we iterate the for loop from 0 to 255.

Now we will create the program which will sum the ascii value of a string.

1. #include <stdio.h>
2. **int** main()
3. {
4. **int** sum=0;  // variable initialization
5. **char** name[20];  // variable initialization
6. **int** i=0;  // variable initialization
7. printf("Enter a name: ");
8. scanf("%s", name);
9. **while**(name[i]!='\0')  // while loop
10. {
11. printf("\nThe ascii value of the character %c is %d", name[i],name[i]);
12. sum=sum+name[i];
13. i++;
14. }
15. printf("\nSum of the ascii value of a string is : %d", sum);
16. **return** 0;
17. }

In the above code, we are taking user input as a string. After taking user input, we execute the **while** loop which adds the ascii value of all the characters of a string and stores it in a '**sum**' variable.

**Output**

