# C - User Input

## Need for User Input in C

Every computer application accepts certain data from the user, performs a predefined process on the same to produce the output. There are no keywords in C that can read user inputs.

The standard library that is bundled with the C compiler includes stdio.h header file, whose library function **scanf()** is most commonly used to **accept user input from the standard input stream**. In addition, the **stdio.h** library also provides other functions for accepting input.

### Example

To understand the need for user input, consider the following C program −

Open Compiler

```
#include <stdio.h>

int main(){
   int price, qty, ttl;

   price = 100;
   qty = 5;
   ttl = price*qty;

   printf("Total: %d", ttl);

   return 0;
}
```

### Output

The above program calculates the total purchase amount by multiplying the price and quantity of an item purchased by a customer. Run the code and check its output −

```
Total: 500
```

For another transaction with different values of price and quantity, you need to edit the program, put the values, then compile and run again. To do this every time is a tedious activity. Instead, there must be a provision to assign values to a variable after the program is run. The **scanf()** function reads the user input during the runtime, and assigns the value to a variable.

## C User Input Function: The scanf()

The C language recognizes the standard input stream as **stdin** and is represented by the standard input device such as a keyboard. C always reads the data from the input stream in the form of characters.

The scanf() function converts the input to a desired data type with appropriate format specifiers.

### Syntax of Scanf()

This is how you would use the scanf() function in C −

```
int scanf(const char *format, &var1, &var2, . . .);
```

The first argument to the scanf() function is a format string. It indicates the data type of the variable in which the user input is to be parsed. It is followed by one or more pointers to the variables. The variable names prefixed by & gives the address of the variable.

### User Input Format Specifiers

Following format specifiers are used in the format string −

| **Format Specifier** | **Type** |
|---|---|
| %c | Character |
| %d | Signed integer |
| %f | Float values |
| %i | Unsigned integer |
| %l or %ld or %li | Long |
| %lf | Double |
| %Lf | Long double |
| %lu | Unsigned int or unsigned long |

| | |
|---|---|
| %lli or %lld | Long long |
| %llu | Unsigned long long |

## Example: User Inputs in C

Going back to the previous example, we shall use the **scanf()** function to accept the value for "price" and "qty", instead of assigning them any fixed value.

Open Compiler

```
#include <stdio.h>

int main(){

   int price, qty, ttl;

   printf("Enter price and quantity: ");
   scanf("%d %d", &price, &qty);

   ttl = price * qty;

   printf("Total : %d", ttl);

   return 0;
}
```

## Output

When the above program is run, C waits for the user to enter the values –

```
Enter price and quantity:
```

When the values are entered and you press Enter, the program proceeds to the subsequent steps.

```
Enter price and quantity: 100 5
Total : 500
```

What is more important is that, for another set of values, you don't need to edit and compile again. Just run the code and the program again waits for the user input. In this way, the program can be run any number of times with different inputs.

```
Enter price and quantity: 150 10
Total : 1500
```

Explore our **latest online courses** and learn new skills at your own pace. Enroll and become a certified expert to boost your career.

## Integer Input

The **%d** format specifier has been defined for signed integer. The following program reads the user input and stores it in the integer variable **num**.

### Example: Integer Input in C

Take a look at the following program code −

Open Compiler

```c
#include <stdio.h>

int main(){

   int num;

   printf("Enter an integer: ");
   scanf("%d", &num);

   printf("You entered an integer: %d", num);

   return 0;
}
```

### Output

Run the code and check its output −

```
Enter an integer: 234
You entered an integer: 234
```

If you enter a non-numeric value, the output will be "0".

The **scanf()** function can read values to one or more variables. While providing the input values, you must separate the consecutive values by a whitespace, a tab or an Enter.

## Example: Multiple Integer Inputs in C

</> Open Compiler

```
#include <stdio.h>

int main(){

   int num1, num2;

   printf("Enter two integers: ");
   scanf("%d %d", &num1, &num2);

   printf("You entered two integers : %d and %d", num1, num2);

   return 0;
}
```

## Output

Run the code and check its output −

```
Enter two integers: 45 57
You entered two integers : 45 and 57
```

or

```
Enter two integers: 45
57
```

## Float Input

For floating point input, you need to use the **%f** format specifier.

## Example: Float Input in C

</> Open Compiler

```
#include <stdio.h>

int main(){

   float num1;

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

   printf("You entered a floating-point number: %f", num1);

   return 0;
}
```

## Output

Run the code and check its output −

```
Enter a number: 34.56
You entered a floating-point number: 34.560001
```

## Example: Integer and Float Inputs in C

The scanf() function may read inputs for different types of variables. In the following program, user input is stored in an integer and a float variable −

</> Open Compiler

```
#include <stdio.h>

int main(){

   int num1;
   float num2;

   printf("Enter two numbers: ");
   scanf("%d %f", &num1, &num2);

   printf("You entered an integer: %d a floating-point number: %6.2f", num1, num2)
   return 0;
}
```

## Output

Run the code and check its output −

```
Enter two numbers: 65 34.5678
You entered an integer: 65 a floating-point number:  34.57
```

## Character Input

The **%c** format specifier reads a single character from the keyboard. However, we must give a blank space before **%c** in the format string. This is because the **%c** conversion specifier won't automatically skip any leading whitespaces.

If there is a stray newline in the input stream (from a previous entry, for example) the **scanf()** call will consume it immediately.

```
scanf(" %c", &c);
```

The blank space in the format string tells scanf to skip the leading whitespace, and the first non-whitespace character will be read with the **%c** conversion specifier.

## Example: Character Input in C

Take a look at the following example −

Open Compiler

```
#include <stdio.h>

int main(){

   char ch;

   printf("Enter a single character: ");
   scanf(" %c", &ch);

   printf("You entered character : %c", ch);
```

```
    return 0;
}
```

## Output

Run the code and check its output −

```
Enter a single character: x
You entered character : x
```

## Example: Multiple Character Inputs in C

The following program reads two characters separated by a whitespace in two char variables.

Open Compiler

```c
#include <stdio.h>

int main(){

   char ch1, ch2;

   printf("Enter two characters: ");
   scanf("%c %c", &ch1, &ch2);

   printf("You entered characters: %c and %c", ch1, ch2);

   return 0;
}
```

## Output

Run the code and check its output −

```
Enter two characters: x y
You entered characters: x and y
```

The **stdio.h** header file also provides the getchar() function. Unlike scanf(), **getchar()** doesn't have a format string. Also, it reads a single key stroke without the Enter key.

## Example: Character Input Using gets()

The following program reads a single key into a char variable −

</> Open Compiler

```
#include <stdio.h>

int main(){

   char ch;

   printf("Enter a character: ");
   ch = getchar();

   puts("You entered: ");
   putchar(ch);

   printf("\nYou entered character: %c", ch);

   return 0;
}
```

### Output

Run the code and check its output −

```
Enter a character: W
You entered:
W
You entered character: W
```

You can also use the unformatted putchar() function to print a single character.

## Example: Reading a Character Sequence

The following program shows how you can read a series of characters till the user presses the Enter key −

</> Open Compiler

```
#include <stdio.h>

int main(){

   char ch;
   char word[10];

   int i = 0;
   printf("Enter characters. End by pressing the Enter key: ");

   while(1){
      ch = getchar();
      word[i] = ch;
      if (ch == '\n')
         break;
      i++;
   }
   printf("\nYou entered the word: %s", word);

   return 0;
}
```

### Output

Run the code and check its output −

```
Enter characters. End by pressing the Enter key: Hello

You entered the word: Hello
```

## String Input

There is also a **%s** format specifier that reads a series of characters into a char array.

### Example: String Input Using scanf()

The following program accepts a string input from the keyboard −

</> Open Compiler

```
#include <stdio.h>

int main(){

   char name[20];

   printf("Enter your name: ");
   scanf("%s", name);

   printf("You entered the name: %s", name);

   return 0;
}
```

## Output

Run the code and check its output −

```
Enter your name: Ravikant
You entered the name: Ravikant
```

C uses the whitespace as the delimiter character. Hence, if you try to input a string that contains a blank space, only the characters before the space are stored as the value.

```
Enter your name: Ravikant Soni
You entered the name: Ravikant
```

The gets() function overcomes this limitation. It is an unformatted string input function. All the characters till you press Enter are stored in the variable.

## Example: String Input Using gets()

Take a look at the following example −

</> Open Compiler

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

int main(){
   char name[20];

   printf("Enter your name: ");
   gets(name);

   printf("You entered the name: %s", name);

   return 0;
}
```

## Output

Run the code and check its output −

```
Enter your name: Ravikant Soni
You entered the name: Ravikant Soni
```

User input is an important aspect of an application in C programming. In this chapter, we explained the usage of formatted and unformatted console input functions, **scanf()**, **getchar()**, and **gets()** with different examples.