**Purpose**

     Prints formatted output.

**Library**

Standard C Library (**libc.a**) or the Standard C Library with 128-Bit long doubles (**libc128.a**)

* **snprintf() in C library**

The snprintf() function formats and stores a series of characters and values in the array buffer.

The snprintf() function with the addition of the n argument, which indicates the maximum number of characters (including at the end of null character) to be written to buffer.

It is defined in <stdio.h>header file.

**Syntax :**

**int snprintf(char \*str, size\_t size, const char \*format, ...);**

**\*str :** is a buffer.

**size :** is the maximum number of bytes(characters) that will be written to the buffer.

**format :** C string that contains a format string that follows the same specifications as format in printf

**... :** the

**optional ( …) arguments** are just the string formats like (“%d”, myint) as seen in printf.

|  |
| --- |
| // C program to demonstrate snprintf()  #include <stdio.h>    int main()  {      char buffer[50];      char\* s = "geeksforgeeks";        // Counting the character and storing      // in buffer using snprintf      int j = snprintf(buffer, 6, "%s\n", s);        // Print the string stored in buffer and      // character count      printf("string:\n%s\ncharacter count = %d\n",                                       buffer, j);        return 0;  } |

Output:

string:

geeks

character count = 14

**2.** Upon successful completion, the **snprintf** subroutine returns the number of bytes written to the *String* parameter (excluding the terminating null byte).

**3**.If output characters are discarded because the output exceeded the *Number* parameter in length, then the **snprintf** subroutine returns the number of bytes that would have been written to the *String* parameter if the *Number* parameter had been large enough (excluding the terminating null byte).

**Practical Example to demonstrate snprintf()**

A practical way to demonstrate snprintf is to implement a **conservative buffer**. We will allocate small amount of memory for a buffer and attempt to place a string into it. We create a larger buffer into it, if necessary. If the string still doesn’t fit, we exit the program.

|  |
| --- |
| // C program to demonstrate snprintf to  // implements conservative buffer  #include <stdio.h>  #include <stdlib.h>    int main()  {      int bufsize = 7;      char\* str = "GeeksforGeeks";      char\* buff = malloc(bufsize);        if (snprintf(buff, bufsize, "%s", str) >= bufsize) {          bufsize \*= 2;          printf("Not enough space. Trying %d bytes\n", bufsize);          free(buff);          buff = malloc(bufsize);            if (snprintf(buff, bufsize, "%s", str) >= bufsize) {              printf("Still not enough space. Aborting\n");              exit(1);          }      }        printf("There was enough space!\n");      printf("buf: %s\n", buff);      return 0;  } |

Output:

Not enough space. Trying 14 bytes

There was enough space!

buf: GeeksforGeeks

However, if **bufsize** was initially set to 5, the output would be the following:  
Output:

Not enough space. Trying 10 bytes

Still not enough space. Aborting

**4.** The **snprintf** subroutine converts, formats, and stores the*Value* parameter values, under control of the *Format* parameter, into consecutive bytes, starting at the address specified by the *String* parameter.

**5.** The **snprintf** subroutine places a null character (\0) at the end. You must ensure that enough storage space is available to contain the formatted string.

**6.** This subroutine provides conversion types to handle code points and **wchar\_t** wide character codes.

**7.**The **snprintf** subroutine is identical to the **sprintf** subroutine with the addition of the *Number* parameter, which states the size of the buffer referred to by the *String* parameter.

**8.**The **snprintf** subroutine allows the insertion of a language-dependent radix character in the output string. The radix character is defined by language-specific data in the **LC\_NUMERIC** category of the program's locale. In the C locale, or in a locale where the radix character is not defined, the radix character defaults to a . (dot).

**9.** The **snprintf** subroutine is unsuccessful if the file specified by the *Stream* parameter is unbuffered or the buffer needs to be flushed and one or more of the following are true:

* **EAGAIN**
* **EBADF**
* **EINTR**
* **EFBIG**

**etc**

* sprintf

The C library function sprintf () is used to store formatted data as astring. You can also say the sprintf () function is used to create strings as output using formatted data.

1.sprintf stands for “String print”. Instead of printing on console, it store output on char buffer which are specified in sprintf.

2. sprintf( ) provides no bounds checking on the array pointed to by buf.it writes to a string instead of writing to a display

3. The array will be overrun if the output generated by sprintf( ) is greater than the array can hold.

**Synta x : sprintf (char \*buf, const char \*format, . . .);**

eg : char str[80];

sprintf(str,"%s %d %c", "one", 2, '3')

4. Upon successful return, these functions return the number of characters printed (excluding the null byte used to end output to strings).If an output error is encountered, a negative value is returned.

5. spritnf is also used as typecasting.

char will convert into int by using sprint.

int i = 12345

char c = "HAI"

sprintf( c, "%d" , 12345)

printf (c)

output: HAI12345

6. The sprintf() function returns the number of bytes that are written in the array, not counting the ending null character.

7. It is defined in <stdio.h>header file.