https://www.gnu.org/software/libc/manual/html\_node/index.html#SEC\_Contents

* The difference between **gets()**and **scanf()**function is that **gets**will accept input from user,until it encounters a new line or EOF ( end of file ).
* **scanf ()**function will accept input from user until it encounters a space or new line.

fgets (name, 100, stdin);

100 is the max length of the buffer. You should adjust it as per your need.

Use:

scanf ("%[^\n]%\*c", name);

The [] is the scanset character. [^\n] tells that while the input is *not* a newline ('\n') take input. Then with the %\*c it reads the newline character from the input buffer (which is not read), and the \* indicates that this read in input is discarded (assignment suppression), as you do not need it, and this newline in the buffer does not create any problem for next inputs that you might take.

**Never use gets(). Because it is impossible to tell without knowing the data in advance how many characters gets() will read, and because gets() will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use fgets() instead.**

**12.9 Line-Oriented Input**

Since many programs interpret input on the basis of lines, it is convenient to have functions to read a line of text from a stream.

Standard C has functions to do this, but they aren’t very safe: null characters and even (for gets) long lines can confuse them. So the GNU C Library provides the nonstandard getline function that makes it easy to read lines reliably.

Another GNU extension, getdelim, generalizes getline. It reads a delimited record, defined as everything through the next occurrence of a specified delimiter character.

All these functions are declared in stdio.h.

Function: *ssize\_t* **getline** *(char \*\*lineptr, size\_t \*n, FILE \*stream)*

Preliminary: | MT-Safe | AS-Unsafe corrupt heap | AC-Unsafe lock corrupt mem | See [POSIX Safety Concepts](https://www.gnu.org/software/libc/manual/html_node/POSIX-Safety-Concepts.html#POSIX-Safety-Concepts).

This function reads an entire line from *stream*, storing the text (including the newline and a terminating null character) in a buffer and storing the buffer address in \**lineptr*.

Before calling getline, you should place in \**lineptr* the address of a buffer \**n* bytes long, allocated with malloc. If this buffer is long enough to hold the line, getline stores the line in this buffer. Otherwise, getline makes the buffer bigger using realloc, storing the new buffer address back in \**lineptr* and the increased size back in \**n*. See [Unconstrained Allocation](https://www.gnu.org/software/libc/manual/html_node/Unconstrained-Allocation.html#Unconstrained-Allocation).

If you set \**lineptr* to a null pointer, and \**n* to zero, before the call, then getline allocates the initial buffer for you by calling malloc. This buffer remains allocated even if getlineencounters errors and is unable to read any bytes.

In either case, when getline returns, \**lineptr* is a char \* which points to the text of the line.

When getline is successful, it returns the number of characters read (including the newline, but not including the terminating null). This value enables you to distinguish null characters that are part of the line from the null character inserted as a terminator.

This function is a GNU extension, but it is the recommended way to read lines from a stream. The alternative standard functions are unreliable.

If an error occurs or end of file is reached without any bytes read, getline returns -1.

Function: *ssize\_t* **getdelim** *(char \*\*lineptr, size\_t \*n, int delimiter, FILE \*stream)*

Preliminary: | MT-Safe | AS-Unsafe corrupt heap | AC-Unsafe lock corrupt mem | See [POSIX Safety Concepts](https://www.gnu.org/software/libc/manual/html_node/POSIX-Safety-Concepts.html#POSIX-Safety-Concepts).

This function is like getline except that the character which tells it to stop reading is not necessarily newline. The argument *delimiter* specifies the delimiter character; getdelim keeps reading until it sees that character (or end of file).

The text is stored in *lineptr*, including the delimiter character and a terminating null. Like getline, getdelim makes *lineptr* bigger if it isn’t big enough.

getline is in fact implemented in terms of getdelim, just like this:

ssize\_t

getline (char \*\*lineptr, size\_t \*n, FILE \*stream)

{

return getdelim (lineptr, n, '\n', stream);

}

Function: *char \** **fgets** *(char \*s, int count, FILE \*stream)*

Preliminary: | MT-Safe | AS-Unsafe corrupt | AC-Unsafe lock corrupt | See [POSIX Safety Concepts](https://www.gnu.org/software/libc/manual/html_node/POSIX-Safety-Concepts.html#POSIX-Safety-Concepts).

The fgets function reads characters from the stream *stream* up to and including a newline character and stores them in the string *s*, adding a null character to mark the end of the string. You must supply *count* characters worth of space in *s*, but the number of characters read is at most *count* - 1. The extra character space is used to hold the null character at the end of the string.

If the system is already at end of file when you call fgets, then the contents of the array *s* are unchanged and a null pointer is returned. A null pointer is also returned if a read error occurs. Otherwise, the return value is the pointer *s*.

**Warning:** If the input data has a null character, you can’t tell. So don’t use fgets unless you know the data cannot contain a null. Don’t use it to read files edited by the user because, if the user inserts a null character, you should either handle it properly or print a clear error message. We recommend using getline instead of fgets.

Function: *wchar\_t \** **fgetws** *(wchar\_t \*ws, int count, FILE \*stream)*

Preliminary: | MT-Safe | AS-Unsafe corrupt | AC-Unsafe lock corrupt | See [POSIX Safety Concepts](https://www.gnu.org/software/libc/manual/html_node/POSIX-Safety-Concepts.html#POSIX-Safety-Concepts).

The fgetws function reads wide characters from the stream *stream* up to and including a newline character and stores them in the string *ws*, adding a null wide character to mark the end of the string. You must supply *count* wide characters worth of space in *ws*, but the number of characters read is at most *count* - 1. The extra character space is used to hold the null wide character at the end of the string.

If the system is already at end of file when you call fgetws, then the contents of the array *ws* are unchanged and a null pointer is returned. A null pointer is also returned if a read error occurs. Otherwise, the return value is the pointer *ws*.

**Warning:** If the input data has a null wide character (which are null bytes in the input stream), you can’t tell. So don’t use fgetws unless you know the data cannot contain a null. Don’t use it to read files edited by the user because, if the user inserts a null character, you should either handle it properly or print a clear error message.

Function: *char \** **fgets\_unlocked** *(char \*s, int count, FILE \*stream)*

Preliminary: | MT-Safe race:stream | AS-Unsafe corrupt | AC-Unsafe corrupt | See [POSIX Safety Concepts](https://www.gnu.org/software/libc/manual/html_node/POSIX-Safety-Concepts.html#POSIX-Safety-Concepts).

The fgets\_unlocked function is equivalent to the fgets function except that it does not implicitly lock the stream.

This function is a GNU extension.

Function: *wchar\_t \** **fgetws\_unlocked** *(wchar\_t \*ws, int count, FILE \*stream)*

Preliminary: | MT-Safe race:stream | AS-Unsafe corrupt | AC-Unsafe corrupt | See [POSIX Safety Concepts](https://www.gnu.org/software/libc/manual/html_node/POSIX-Safety-Concepts.html#POSIX-Safety-Concepts).

The fgetws\_unlocked function is equivalent to the fgetws function except that it does not implicitly lock the stream.

This function is a GNU extension.

Deprecated function: *char \** **gets** *(char \*s)*

Preliminary: | MT-Safe | AS-Unsafe corrupt | AC-Unsafe lock corrupt | See [POSIX Safety Concepts](https://www.gnu.org/software/libc/manual/html_node/POSIX-Safety-Concepts.html#POSIX-Safety-Concepts).

The function gets reads characters from the stream stdin up to the next newline character, and stores them in the string *s*. The newline character is discarded (note that this differs from the behavior of fgets, which copies the newline character into the string). If gets encounters a read error or end-of-file, it returns a null pointer; otherwise it returns *s*.

**Warning:** The gets function is **very dangerous** because it provides no protection against overflowing the string *s*. The GNU C Library includes it for compatibility only. You should **always** use fgets or getline instead. To remind you of this, the linker (if using GNU ld) will issue a warning whenever you use gets.

void interactivemode ()

{

printf ("Entering interactive mode\n");

printf ("This mode is currently useles, all you type will be parsed then displayed\n");

size\_t len=0;

int delim = (int) '\n';

while (1) {

char \*text = NULL;

getdelim(&text, &len, delim, stdin);

if (len == -1) {

printf ("error, no input\n");

continue;

}

printf("entered: %s\n",text);

statement = parsestatement (text);

displaystatement (statement);

free(text); // double check man page, think you need this to prevent memory leak.

}

}

The standard input function, cin, ignores space, tabs, and newlines. In most cases, you want the invisible characters the user enters to be stored in a string as well. Or you want to get the entire line of data from the input stream or file stream. In these situations, you can use getline() functions. Their prototypes look something like this:

istream & getline(istream & is, string & s, char delim)  
istream & getline(istream & is, string & s)

# Two versions of getline()

In the older version of c++ when string (derived datatypes ) weren’t introduced, at that time it takes the input as string in the form of array of characters

Syntax :

1. **char** string1[10000]
2. cin.getline(string1,100000);

here getline acts as a function of object c under istream class.

but now in updated version , string (derived datatype) has been introduced in c++ and we can input string direct as

Syntax:

1. **string** string2
2. getline(cin,string2)

the second version is preferred now days ,because it eliminates the limits of array, you don’t have to worry about memory allocation.

#include<iostream>

#include<bits/stdc++.h>

**using** **namespace** std;

**int** main()

{

/\*When you are Dealing with array of characters as a String\*/

**char** String1[1000];

cout<<"Enter the String"<<endl<<endl;

cin.getline(String1,1000);

cout<<"You entered :"<<endl;

cout<<String1;

cout<<endl;

/\*When you are dealing with string using stdc++ liberary\*/

**string** string2;

cout<<"Enter the String "<<endl<<endl;

getline(cin,string2);

cout<<"You enterd : "<<endl;

cout<<string2;

cout<<endl;

**return** 0;

}