scanf

```
<cstdio>
int  scanf ( const char * format, ... );
```

Read formatted data from stdin

Reads data from `stdin` and stores them according to the parameter *format* into the locations pointed by the additional arguments. The additional arguments should point to already allocated objects of the type specified by their corresponding format tag within the *format* string.

Parameters

format

C string that contains one or more of the following items:

- **Whitespace character:** the function will read and ignore any whitespace characters (this includes blank spaces and the newline and tab characters) which are encountered before the next non-whitespace character. This includes any quantity of whitespace characters, or none.
- **Non-whitespace character, except percentage signs (%):** Any character that is not either a whitespace character (blank, newline or tab) or part of a format specifier (which begin with a % character) causes the function to read the next character from `stdin`, compare it to this non-whitespace character and if it matches, it is discarded and the function continues with the next character of *format*. If the character does not match, the function fails, returning and leaving subsequent characters of `stdin` unread.
- **Format specifiers:** A sequence formed by an initial percentage sign (%) indicates a format specifier, which is used to specify the type and format of the data to be retrieved from `stdin` and stored in the locations pointed by the additional arguments. A format specifier follows this prototype:

  %[*][width][modifiers]type

  where:

*	An optional starting asterisk indicates that the data is to be retrieved from `stdin` but ignored, i.e. it is not stored in the corresponding argument.
width	Specifies the maximum number of characters to be read in the current reading operation
modifiers	Specifies a size different from `int` (in the case of `d`, `i` and `n`), `unsigned int` (in the case of `o`, `u` and `x`) or `float` (in the case of `e`, `f` and `g`) for the data pointed by the corresponding additional argument: **h :** `short int` (for `d`, `i` and `n`), or `unsigned short int` (for `o`, `u` and `x`) **l :** `long int` (for `d`, `i` and `n`), or `unsigned long int` (for `o`, `u` and `x`), or `double` (for `e`, `f` and `g`) **L :** `long double` (for `e`, `f` and `g`)
type	A character specifying the type of data to be read and how it is expected to be read. See next table.

- **scanf type specifiers:**

type	Qualifying Input	Type of argument
c	**Single character:** Reads the next character. If a *width* different from 1 is specified, the function reads *width* characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.	`char *`
d	**Decimal integer:** Number optionally preceded with a `+` or `-` sign.	`int *`
e,E,f,g,G	**Floating point:** Decimal number containing a decimal point, optionally preceded by a `+` or `-` sign and optionally folowed by the `e` or `E` character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4	`float *`
o	**Octal integer**.	`int *`
s	**String of characters**. This will read subsequent characters until a whitespace is found (whitespace characters are considered to be blank, newline and tab).	`char *`
u	Unsigned decimal integer.	`unsigned int *`
x,X	Hexadecimal integer.	`int *`

additional arguments

The function expects a sequence of references as additional arguments, each one pointing to an object of the type specified by their corresponding %-tag within the *format* string, in the same order.
For each format specifier in the *format* string that retrieves data, an additional argument should be specified.
These arguments are expected to be references (pointers): if you want to store the result of a `fscanf` operation on a regular variable you should precede its identifier with the *reference operator*, i.e. an ampersand sign (&), like in:

```
int n;
scanf ("%d",&n);
```

Return Value

On success, the function returns the number of items succesfully read. This count can match the expected number of readings or fewer, even zero, if a matching failure happens.
In the case of an input failure before any data could be successfully read, EOF is returned.