

Input and Output Functions

Apart from 'cin' and 'cout', there are many other functions we can use to perform input/output operations.

WE'LL COVER THE FOLLOWING ^

- Input
 - Formatted input
 - Unformatted input
- Output
- Further information

Input

In C++ we can read from the input stream in two different ways: Formatted with the extractor `>>` and unformatted with explicit methods.

Formatted input

The extraction operator `>>`:

- is predefined for all built-in types and strings.
- can be implemented for [user-defined data types].
(<https://www.educative.io/collection/page/10370001/5128982204776448/5928180936343552>)
- can be configured by format specifiers.

 **`std::cin` ignores, by default, leading whitespace**

```
#include <iostream>
//...
int a, b;
std::cout << "Two natural numbers: " << std::endl;
std::cin >> a >> b; // < 2000 11>
```

```
std::cout << "a: " << a << " b: " << b;
```

Unformatted input

There are many methods that can be used for the unformatted input from an input stream `is`.

Method	Description
<code>is.get(ch)</code>	Reads one character into <code>ch</code> .
<code>is.get(buf, num)</code>	Reads, at most, <code>num</code> characters into the buffer <code>buf</code> .
<code>is.getline(buf, num[, delim])</code>	Reads, at most, <code>num</code> characters into the buffer <code>buf</code> . Optionally uses the line-delimiter <code>delim</code> (default <code>\n</code>).
<code>is.gcount()</code>	Returns the number of characters that were last extracted from <code>is</code> by an unformatted operation.
<code>is.ignore(streamsize sz= 1, int delim= end-of-file)</code>	Ignores <code>sz</code> characters until <code>delim</code> .
<code>is.peek()</code>	Gets one character from <code>is</code> without consuming it.
<code>is.unget()</code>	Pushes the last read character back to <code>is</code> .
<code>is.putback(ch)</code>	Pushes the character <code>ch</code> onto the stream <code>is</code> .

Unformatted input from an input stream

🔑 `std::string` has a `getline` function

The `getline` function of `std::string` has a big advantage above the `getline` function of the `istream`. The `std::string` automatically takes care of its memory. On the contrary, we have to reserve the memory for the buffer `buf` in the `is.get(buf, num)` function.

```
#include <iostream>
#include <string>

int main(){

    std::cout << std::endl;

    std::string line;
    std::cout << "Write a line: " << std::endl;
    std::getline(std::cin, line);
    std::cout << line << std::endl;

    std::cout << std::endl;

    std::cout << "Write numbers, separated by;" << std::endl;
    while ( std::getline(std::cin, line, ';') ) {
        std::cout << line << std::endl;
    }

    std::cout << std::endl;
}
```



Unformatted input

Output

We can push characters with the insert operator `<<` onto the output stream.

The insert operator `<<`

- is predefined for all built-in types and strings.
- can be implemented for [user-defined data types].
(<https://www.educative.io/collection/page/10370001/5128982204776448/5928180936343552>)
- can be adjusted by format specifiers.

can be adjusted by format specifiers.

Further information

- `getline` function
- user-defined data types

Let's solve an exercise in the next lesson.