How to read until EOF from cin in C++

I am coding a program that reads data directly from user input and was wondering how could I (without loops) read all data until EOF from standard input. I was considering using `cin.get( input, '\0' )` but `'\0'` is not really the EOF character, that just reads until EOF or `'\0'`, whichever comes first.

Or is using loops the only way to do it? If so, what is the best way?

c++ input iostream

edited Sep 15 '12 at 16:29

community wiki
5 revs, 3 users 60%
Guille

10 Answers

The only way you can read a variable amount of data from stdin is using loops. I've always found that the `std::getline()` function works very well:

```
std::string line;
while (std::getline(std::cin, line))
{
    std::cout << line << std::endl;
}
```

By default getline() reads until a newline. You can specify an alternative termination character, but EOF is not itself a character so you cannot simply make one call to getline().

edited Aug 27 '13 at 13:51
user283145

answered Oct 14 '08 at 17:16
trotterdylan
639 4 3

4 Something to be careful of here is when you're reading from files that have been piped in via stdin, is that this and most of the answers given will append an extra line feed on the end of your input. Not all files end with a line feed, yet each iteration of the loop appends "std::endl" to the stream. This may not be an issue in most cases, but if file integrity is an issue, this may come back to bite you. – Zoccadoum Mar 29 '16 at 0:41

This should not be the most upvoted answer. See the below community wiki answer. Also, maybe see this question: stackoverflow.com/questions/3203452/… – loxaxs Aug 16 at 10:21

You can do it without explicit loops by using stream iterators. I'm sure that it uses some kind of loop internally.

```
#include <string>
#include <iostream>
#include <istream>
#include <ostream>
#include <iterator>

int main()
{
// don't skip the whitespace while reading
  std::cin >> std::noskipws;

// use stream iterators to copy the stream to a string
  std::istream_iterator<char> it(std::cin);
  std::istream_iterator<char> end;
  std::string results(it, end);

  std::cout << results;
}
```

answered Oct 14 '08 at 17:35
KeithB
13.2k 2 29 41

2 Nice. As for "I'm sure that it uses some kind of loop internally" - any solution would. – Michael Burr Oct 14

'08 at 17:39

This seems to remove newline characters from input, even if they occur right in the middle. – Multisync Sep 7 at 8:44

Using loops:

```
#include <iostream>
using namespace std;
...
// numbers
int n;
while (cin >> n)
{
   ...
}
// lines
string line;
while (getline(cin, line))
{
   ...
}
// characters
char c;
while (cin.get(c))
{
   ...
}
```

resource

edited Oct 14 '08 at 18:00

answered Oct 14 '08 at 17:43

Degvik
1,261 5 16 20

After researching KeithB's solution using `std::istream_iterator`, I discovered the `std:istreambuf_iterator`.

Test program to read all piped input into a string, then write it out again:

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

int main()
{
  std::istreambuf_iterator<char> begin(std::cin), end;
  std::string s(begin, end);
  std::cout << s;
}
```

edited Feb 19 at 16:49

community wiki
2 revs, 2 users 93%
Richard Smith

1 This should probably be the canonical answer. – Kerrek SB Feb 19 at 16:43

Downvoted. So what are the pros or cons compared to KeithB's solution? Yes, you are using an `std::istreambuf_iterator` instead of an `std::istream_iterator`, but what for? – Multisync Sep 7 at 8:49

Probable simplest and generally efficient:

```
#include <iostream>
int main()
{
    std::cout << std::cin.rdbuf();
}
```

If needed, use stream of other types like `std::ostringstream` as buffer instead of standard output stream here.

answered May 18 '15 at 15:04

community wiki
FrankHB

you guys already know, but may be helpful to some: `std::ostringstream std_input; std_input << std::cin.rdbuf(); std::string s = std_input.str()`. you have all the input in `s` (be careful with `std::string` if the input is too big) – ribamar Dec 1 '15 at 17:42

You can use the std::istream::getline() (or preferably the version that works on std::string) function to get an entire line. Both have versions that allow you to specify the delimiter (end of line character). The default for the string version is '\n'.

edited Oct 14 '08 at 17:45

answered Oct 14 '08 at 17:14
luke
24.6k 6 51 76

Sad side note: I decided to use C++ IO to be consistent with boost based code. From answers to this ques[illegible] line)) . Using g++ version 4.5.3 (-O3) in c[illegible] sual C++ 2010 (/O2) made it 40 MB/s [illegible]

After rewriting the IO to pure C `while (fgets(buf, 100, stdin))` the throughput jumped to 90 MB/s in both tested compilers. That makes a difference for any input bigger than 10 MB...

answered Aug 3 '12 at 10:10

community wiki
liborm

No need to rewrite your code to pure C, just add a `std::ios::sync_with_stdio(false);` before your while-loop. `cin.tie(NULL);` may also help if you interleave reading and writing. – R D Nov 19 at 21:52

```
while(std::cin) {
 // do something
}
```

edited Aug 10 '12 at 16:24

community wiki
2 revs, 2 users 57%
Bryan

---

6 This is a terrible disaster of an answer, since it's virtually guaranteed to result in wrong code. – Kerrek SB Feb 19 at 16:43

Wait, am I understanding you correctly? You're using cin for keyboard input, and you want to stop reading input when the user enters the EOF character? Why would the user ever type in the EOF character? Or did you mean you want to stop reading from a file at the EOF?

If you're actually trying to use cin to read an EOF character, then why not just specify the EOF as the delimiter?

```
// Needed headers: iostream

char buffer[256];
cin.get( buffer, '\x1A' );
```

If you mean to stop reading from a file at the EOF, then just use getline and once again specify the EOF as the delimiter.

```
// Needed headers: iostream, string, and fstream

string buffer;

    ifstream fin;
    fin.open("test.txt");
    if(fin.is_open()) {
        getline(fin,buffer,'\x1A');

        fin.close();
    }
```

edited Aug 10 '12 at 17:18

community wiki
3 revs
uberwulu

One option is to a use a container, e.g.

```
std::vector<char> data;
```

and *redirect* all input into this collection until `EOF` is received, i.e.

```
std::copy(std::istream_iterator<char>(std::cin),
    std::istream_iterator<char>(),
    std::back_inserter(data));
```

However, the used container might need to reallocate memory too often, or you will end with a `std::bad_alloc` exception when your system gets out of memory. In order to solve these problems, you could *reserve* a fixed amount `N` of elements and process these amount of elements in isolation, i.e.

```
data.reserve(N);
while (/*some condition is met*/)
{
    std::copy_n(std::istream_iterator<char>(std::cin),
        N,
        std::back_inserter(data));

    /* process data */

    data.clear();
}
```

answered Dec 18 '14 at 23:13

community wiki
0xbadf00d