Reading from text file until EOF repeats last line

The following **C++** code uses a **ifstream** object to read integers from a text file (which has one number per line) until it hits **EOF**. Why does it read the integer on the last line twice? How to fix this?

Code:

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ifstream iFile("input.txt");        // input.txt has integers, one per line

    while (!iFile.eof())
    {
        int x;
        iFile >> x;
        cerr << x << endl;
    }

    return 0;
}
```

input.txt:

```
10
20
30
```

Output:

```
10
20
30
30
```

Note: I've skipped all error checking code to keep the code snippet small. The above behaviour is seen on Windows (Visual C++), cygwin (gcc) and Linux (gcc).

c++ iostream fstream

edited Jul 9 '11 at 22:22

Lightness Races in Orbit
142k ● 26 ● 205 ● 390

asked Aug 22 '08 at 2:21

Ashwin
19.4k ● 32 ● 125 ● 202

7 Answers

Just follow closely the chain of events.

- Grab 10
- Grab 20
- Grab 30
- Grab EOF

Look at the second-to-last iteration. You grabbed 30, then carried on to check for EOF. You haven't reached EOF because the EOF mark hasn't been read yet ("binarically" speaking, its conceptual location is just after the 30 line). Therefore you carry on to the next iteration. x is still 30 from previous iteration. Now you read from the stream and you get EOF. x remains 30 and the ios::eofbit is raised. You output to stderr x (which is 30, just like in the previous iteration). Next you check for EOF in the loop condition, and this time you're out of the loop.

Try this:

```
while (true) {
    int x;
    iFile >> x;
    if( iFile.eof() ) break;
    cerr << x << endl;
}
```

By the way, there is another bug in your code. Did you ever try to run it on an empty file? The behaviour you get is for the exact same reason.

edited Aug 22 '08 at 2:57

answered Aug 22 '08 at 2:50

wilhelmtell

27.6k ● 9 ● 61 ● 103

22 use 'while(iFile >> x)'. This reads the integer and returns the stream. When a stream is used as bool value it checks to see if the stream is valid. Valid means eof() and bad() are both false. see stackoverflow.com/questions/21647/… – Crappy Experience Bye Sep 23 '08 at 7:48

1 Same spirit as up comment: instead of `while(true)` , it seems better to write `while(file.good())` – PHF Apr 13 '12 at 15:14

I like this example, which for now, leaves out the check which you could add inside the while block:

```
ifstream iFile("input.txt");        // input.txt has integers, one per line
int x;

while (iFile >> x)
{
    cerr << x << endl;
}
```

Not sure how safe it is...

answered Aug 22 '08 at 2:59

Patrick Loz

327 ● 2 ● 8

What if 0 is a valid value, like x==0? – harryngh Jul 4 '14 at 11:21

There's an alternative approach to this:

```
#include <iterator>
#include <algorithm>

// ...

    copy(istream_iterator<int>(iFile), istream_iterator<int>(),
         ostream_iterator<int>(cerr, "\n"));
```

edited Aug 22 '08 at 4:52

answered Aug 22 '08 at 4:46

wilhelmtell

27.6k ● 9 ● 61 ● 103

Without to much modifications of the original code, it could become :

```
while (!iFile.eof())
{
    int x;
    iFile >> x;
    if (!iFile) break;
    cerr << x << endl;
}
```

but I prefer the two other solutions above in general.

answered Feb 12 '13 at 16:17

Solostaran14

929 ● 8 ● 14

```
int x;
ifile >> x

while (!iFile.eof())
{
    cerr << x << endl;
    iFile >> x;
}
```

answered Apr 11 '13 at 20:29

user1384482

9 ● 1

The EOF pattern needs a prime read to 'bootstrap' the EOF checking process. Consider the empty file will not initially have its EOF set until the first read. The prime read will catch the EOF in this instance and properly skip the loop completely.

What you need to remember here is that you don't get the EOF until the first attempt to read past the available data of the file. Reading the exact amount of data will not flag the EOF.

I should point out if the file was empty your given code would have printed since the EOF will have prevented a value from being set to x on entry into the loop.

- 0

So add a prime read and move the loop's read to the end:

```
int x;

iFile >> x; // prime read here
while (!iFile.eof()) {
    cerr << x << endl;
    iFile >> x;
}
```

answered Dec 3 '14 at 13:56

Brian Jack

310 ● 2 ● 8

At the end of the last line, you have a new line character, which is not read by >> operator and it is not an end of file. Please, make an experiment and delete the new line (thelast character in file) - you will not get the duplication. To have a flexible code and avoid unwanted effects just apply any solution given by other users.

answered Dec 20 '14 at 22:53

Raf

31 ● 3
