Reading and writing binary file

Asked 12 years, 7 months ago Modified 2 years, 8 months ago Viewed 415k times

151

I'm trying to write code to read a binary file into a buffer, then write the buffer to another file. I have the following code, but the buffer only stores a couple of ASCII characters from the first line in the file and nothing else.

```
int length;
char * buffer;

ifstream is;
is.open ("C:\\Final.gif", ios::binary );
// get length of file:
is.seekg (0, ios::end);
length = is.tellg();
is.seekg (0, ios::beg);
// allocate memory:
buffer = new char [length];
// read data as a block:
is.read (buffer,length);
is.close();

FILE *pFile;
pFile = fopen ("C:\\myfile.gif", "w");
fwrite (buffer , 1 , sizeof(buffer) , pFile );
```

c++ file binary buffer

Share Follow

edited Apr 22, 2016 at 17:11
gsamaras
72.1k 46 188 306

asked Mar 24, 2011 at 14:00
nf313743
4,129 8 48 63

37 You should decide to use iostream or C file handling. Please do not use both. – frast Mar 24, 2011 at 14:03

2 There is a mistake in the above code regarding the buffer variable. It's type should be `unsigned char` and the allocation should be `buffer = new unsigned char[length + 1]` and then `buffer[length] = '\0'` . I know that the question was posted many years ago, but nobody has written about this. – Raluca Pandaru Mar 28, 2022 at 5:10

1 @RalucaPandaru, given the input file is a GIF and explicitly read as `ios::binary` , adding a zero-terminator as you suggested makes no sense. Also difference between using a `char` or `unsigned char` does not make much difference here, as the code is not trying to interpret the file content. What is wrong with the above code is the `sizeof(buffer)` is 4 or 8 (32/64-bit pointer). So it always writes 4/8 bytes

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email Sign up with Google Sign up with GitHub Sign up with Facebook

Highest score (default)

8 Answers

Sorted by:

240

If you want to do this the C++ way, do it like this:

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

int main()
{
    std::ifstream input( "C:\\Final.gif", std::ios::binary );
    std::ofstream output( "C:\\myfile.gif", std::ios::binary );

    std::copy(
        std::istreambuf_iterator<char>(input),
        std::istreambuf_iterator<char>( ),
        std::ostreambuf_iterator<char>(output));
}
```

If you need that data in a buffer to modify it or something, do this:

```
#include <fstream>
#include <iterator>
#include <vector>

int main()
{
    std::ifstream input( "C:\\Final.gif", std::ios::binary );

    // copies all data into buffer
    std::vector<unsigned char> buffer(std::istreambuf_iterator<char>(input), {});

}
```

Share Follow

edited Nov 1, 2018 at 11:52
Evgeny Yashin
58 9

answered Mar 24, 2011 at 14:19
Björn Pollex
75.5k 28 201 283

6 What if I want copy only some segment of data to buffer. How I can do it? Let say 1024 bytes. – likern Jul 10, 2014 at 15:45

9 @Mikhail Here you can find some benchmarking. – Paolo M Feb 1, 2016 at 16:06

3 AFAIK, binary files sometimes contain unreadable char, in the fact, they are not char at all. Is this code safe for reading non-text base file? My knowledge is short in this range :) – Andiana Nov 8, 2016 at 15:09

8 so-called `char` is used in C/C++ to store bytes (and have been for the last 40 years). it's safe to do so, as long as you don't try to actually **USE** that data as characters (don't use strlen() on it, don't print it to

Join Stack Overflow to find the best answer to your technical question, help others answer theirs.

Sign up with email

2 @DavidTran Can't say without knowing more - this seems like you should create a minimal example that reproduces the issue, and then post a question. – Björn Pollex May 8, 2019 at 6:19

17

Here is a short example, the C++ way using `rdbuf`. I got this from the web. I can't find my original source on this:

```
#include <fstream>
#include <iostream>

int main ()
{
  std::ifstream f1 ("C:\\me.txt",std::fstream::binary);

  std::ofstream f2 ("C:\\me2.doc",std::fstream::trunc|std::fstream::binary);

  f2<<f1.rdbuf();

  return 0;
}
```

Share Follow edited Jan 13, 2017 at 18:16 answered Mar 24, 2011 at 17:07

Thomas Matthews
57k 17 99 154

12 The best, non portable, method is to let the OS copy your file. After all, that is part of what it does for a living; no need to *reinvent the wheel*. – Thomas Matthews Mar 24, 2011 at 17:08

15

```
sizeof(buffer) == sizeof(char*)
```

Use length instead.

Also, better to use `fopen` with " `wb` "....

Share Follow edited Jul 14, 2013 at 17:02 answered Mar 24, 2011 at 14:03

NullPoiиteя
56.7k 22 126 143

Alexey Sudachen
374 2 5

Can't use `buffer.length()` for buffer may have NULL values inside it thereby defeating the purpose of strlen/length(). – John Greene Aug 30, 2017 at 17:06

Better to use `sizeof(buffer)`. – John Greene Aug 30, 2017 at 18:50

10

Share Follow

answered Mar 24, 2011 at 14:01
jcoder
29.6k 19 89 130

6

You should pass length into fwrite instead of sizeof(buffer).

Share Follow

answered Mar 24, 2011 at 14:02
retrodrone
5,850 9 39 65

4

Here is implementation of standard C++ 14 using **vectors** and **tuples** to Read and Write `Text,Binary and Hex files`.

Snippet code :

```
try {
if (file_type == BINARY_FILE) {

    /*Open the stream in binary mode.*/
    std::ifstream bin_file(file_name, std::ios::binary);

    if (bin_file.good()) {
        /*Read Binary data using streambuffer iterators.*/
        std::vector<uint8_t> v_buf((std::istreambuf_iterator<char>(bin_file)),
(std::istreambuf_iterator<char>()));
        vec_buf = v_buf;
        bin_file.close();
    }

    else {
        throw std::exception();
    }

}

else if (file_type == ASCII_FILE) {

    /*Open the stream in default mode.*/
    std::ifstream ascii_file(file_name);
}
```

Full Source code can be found [here]

Share Follow

answered Feb 4, 2021 at 2:10

HeavenHM
946 1 13 22

-1

It can be done with simple commands in the following snippet.

Copies the whole file of any size. No size constraint!

Just use this. Tested And Working!!

```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
  ifstream infile;
  infile.open("source.pdf",ios::binary|ios::in);

  ofstream outfile;
```

```
        outfile.write((char *)&buffer,sizeof(buffer));
```

Having a smaller buffer size would be helpful in copying tiny files. Even "char buffer[2]" would do the job.

Share Follow edited Sep 9, 2018 at 16:05 answered Nov 2, 2014 at 15:33
iMajetyHK
157 1 1 7

9 And what if file size isn't multiple of buffer size? Moreover, why do you have to declare your buffer as `int[]` instead of `char[]` ? – firegurafiku May 13, 2016 at 11:33

I already mentioned it works with `char[]` too and files of any size which means there's no condition that file size should be a multiple of buffer size. – iMajetyHK Sep 9, 2018 at 16:04

2 The fact that you said it works does not mean it works. The fact that it does not work means it does not work. – nunojpg May 27, 2020 at 11:42

The least you could do is to change 'int buffer[2]' to 'char buffer[1]' to make things work without changing the code to fix bugs. – Ruud van Gaal Aug 23, 2021 at 9:00

-2

There is a much simpler way. This does not care if it is binary or text file.

Use noskipws.

```
char buf[SZ];
ifstream f("file");
int i;
for(i=0; f >> noskipws >> buffer[i]; i++);
ofstream f2("writeto");
for(int j=0; j < i; j++) f2 << noskipws << buffer[j];
```

Or you can just use string instead of the buffer.

```
string s; char c;
ifstream f("image.jpg");
while(f >> noskipws >> c) s += c;
ofstream f2("copy.jpg");
f2 << s;
```

Share Follow edited Nov 8, 2020 at 14:25 answered May 10, 2018 at 1:14

Zeta
923 10 24

You might want to elaborate on this to make it more understandable – jvh Oct 5, 2020 at 11:15