Tutorial 2: File Input/Output

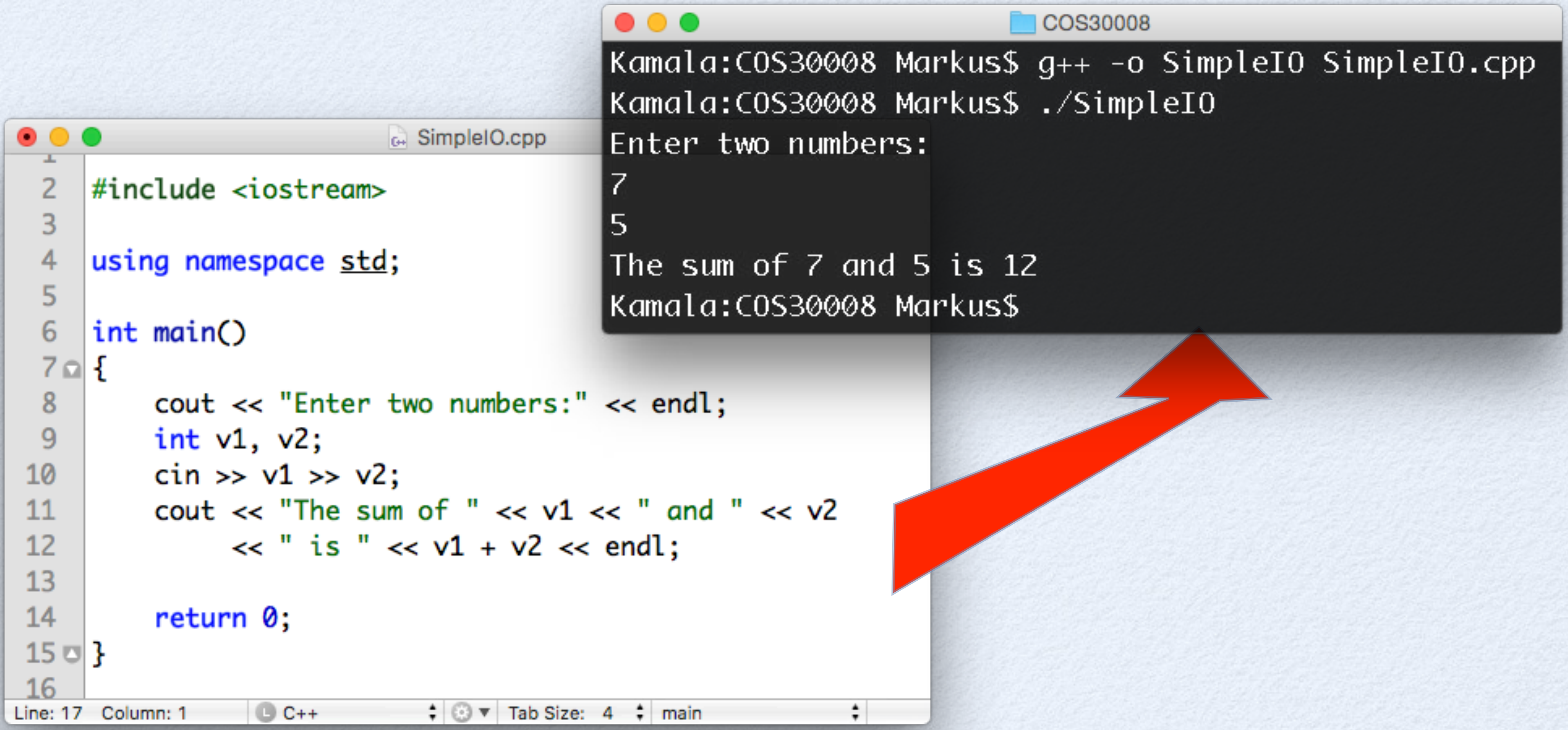
Overview

- Standard I/O and Files

References

- Gary J. Bronson: C++ for Engineers and Scientists. 3rd Edition. Thomson (2010)
- Stanley B. Lippman, Josée Lajoie, and Barbara E. Moo: C++ Primer. 5th Edition. Addison-Wesley (2013)
- Gary J. Bronson: Object-Oriented Program Development Using C++ - A Class-Centered Approach. Thomson (2006)

Command Line Input and Output



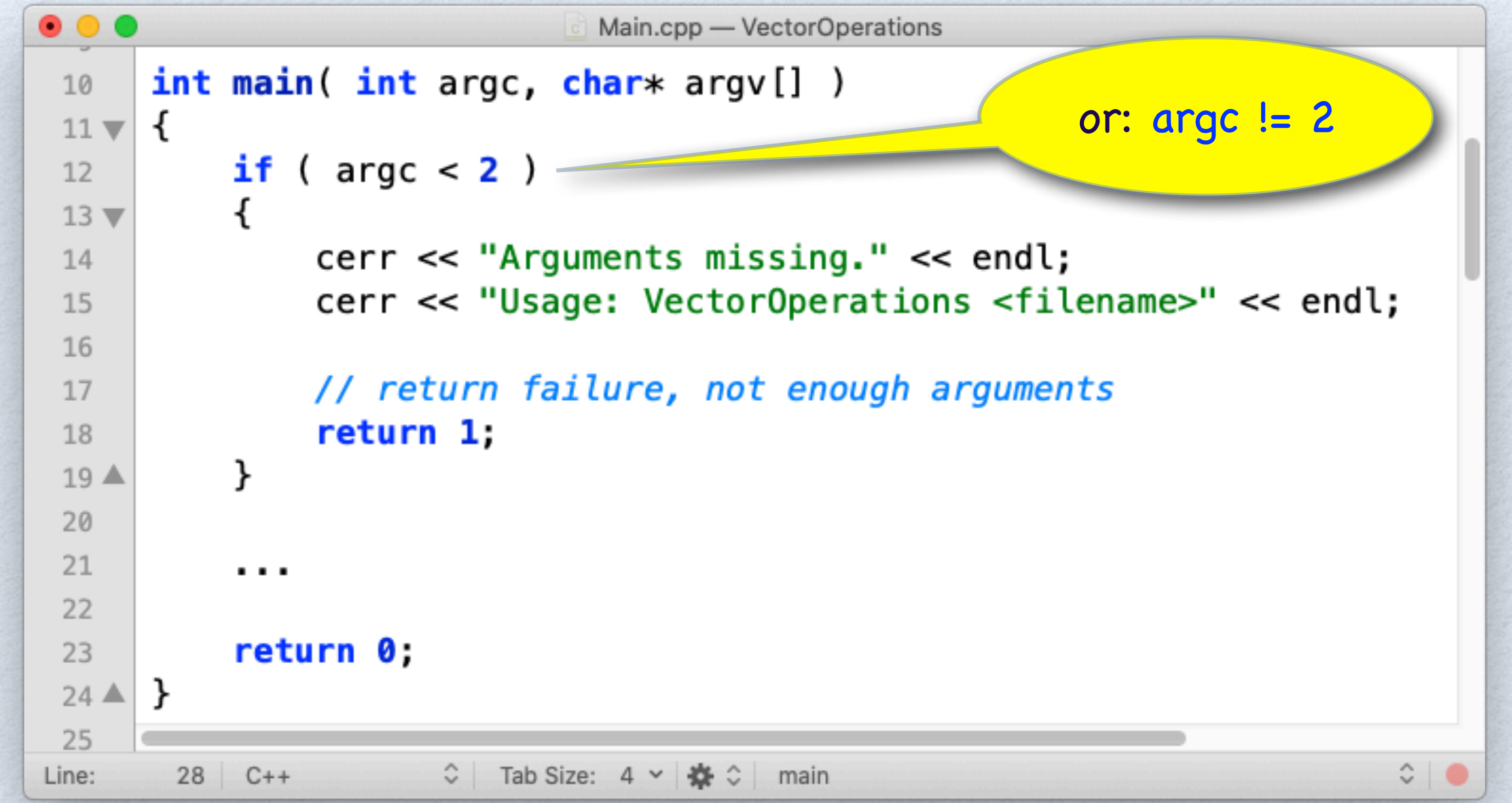
```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Enter two numbers:" << endl;
    int v1, v2;
    cin >> v1 >> v2;
    cout << "The sum of " << v1 << " and " << v2
         << " is " << v1 + v2 << endl;

    return 0;
}
```

```
Kamala:COS30008 Markus$ g++ -o SimpleIO SimpleIO.cpp
Kamala:COS30008 Markus$ ./SimpleIO
Enter two numbers:
7
5
The sum of 7 and 5 is 12
Kamala:COS30008 Markus$
```

I/O Media

- Streams can be associated with
  - Physical devices (e.g., console - cin, cout)
  - Files (e.g., coefficients.txt, sales.dbf)
  - Structured storage (e.g., int values[10])

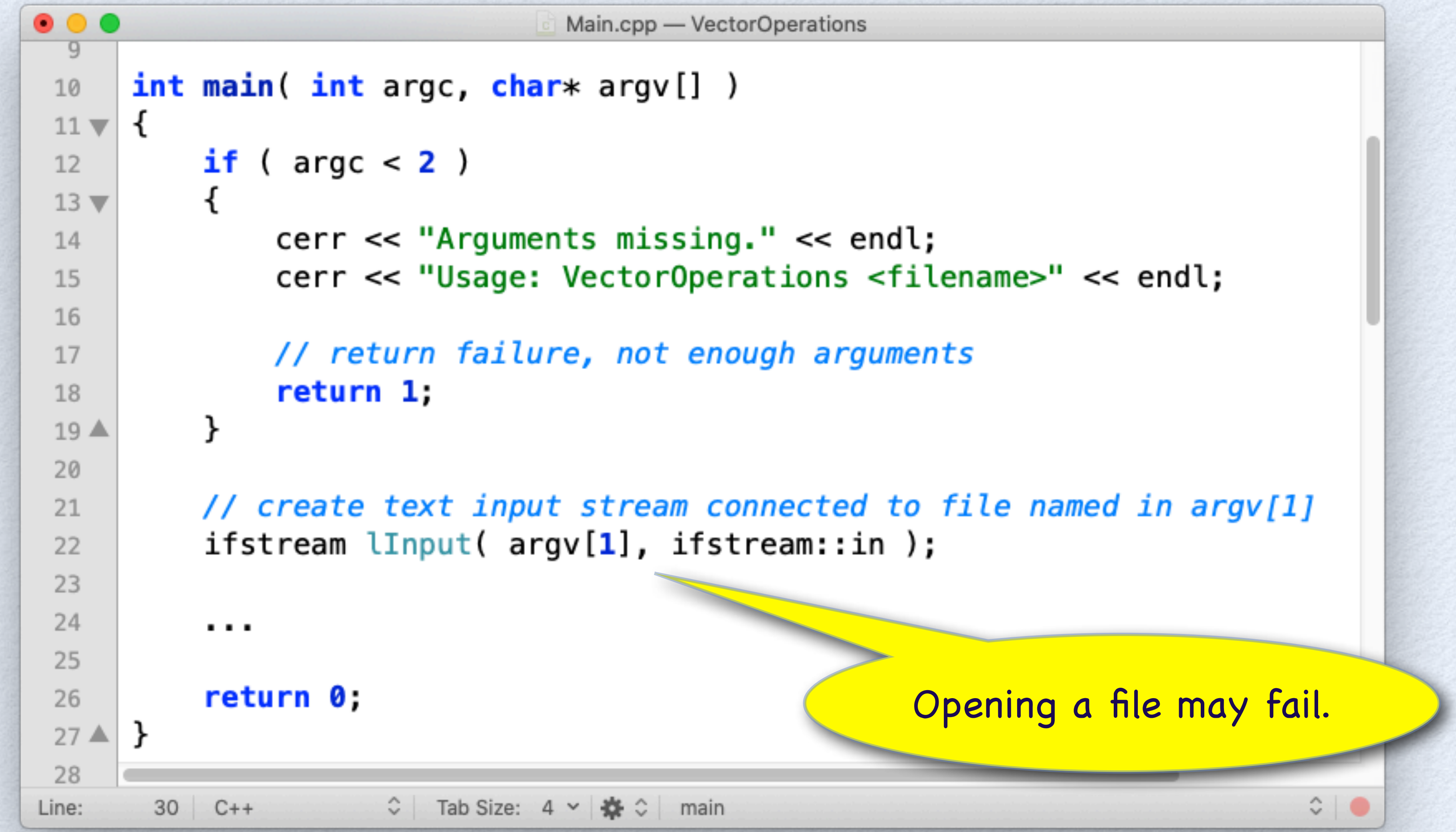
Working With Files - Program Arguments



```
int main( int argc, char* argv[] )
{
    if ( argc < 2 )
    {
        cerr << "Arguments missing." << endl;
        cerr << "Usage: VectorOperations <filename>" << endl;

        // return failure, not enough arguments
        return 1;
    }

    ...

    return 0;
}
```

- We can pass the names of the files our program needs to work with through the command line arguments.

Set up an Input File



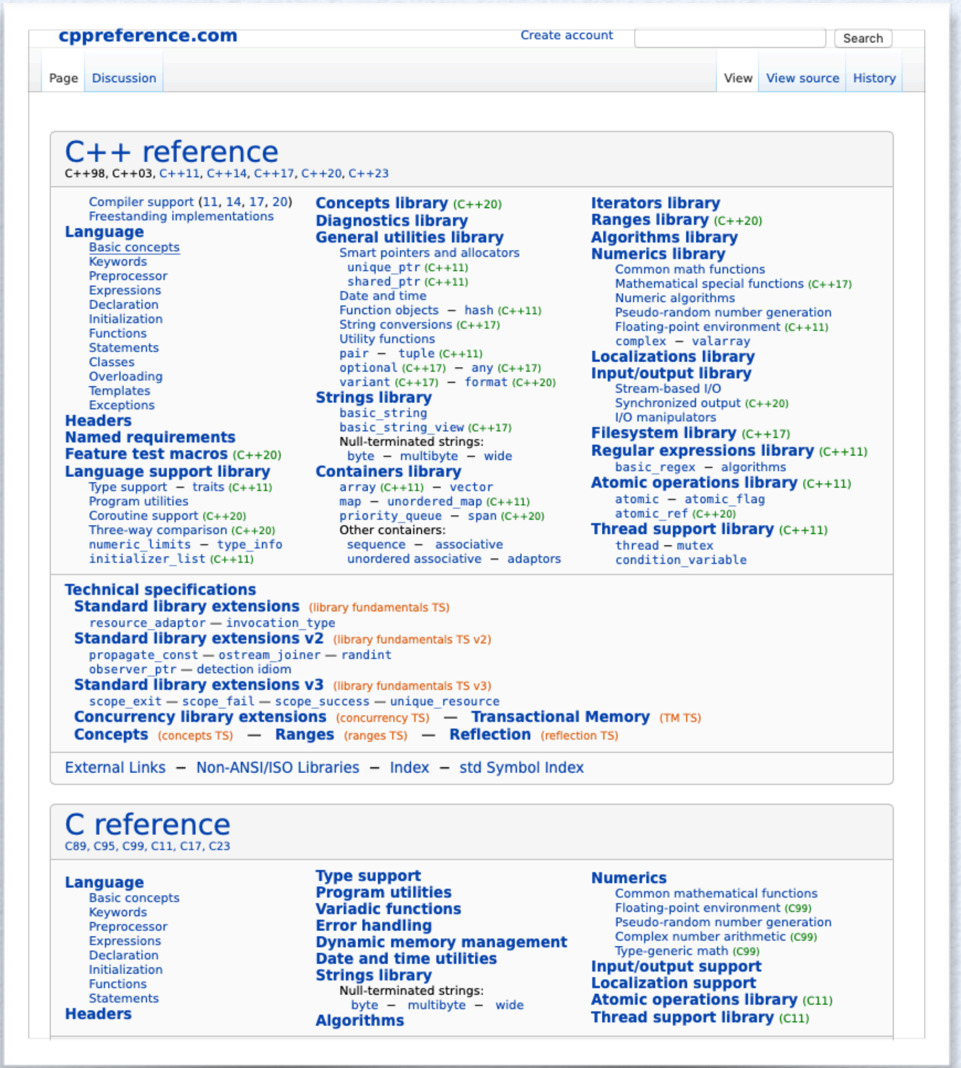
Main.cpp — VectorOperations

```cpp
int main( int argc, char* argv[] )
{
    if ( argc < 2 )
    {
        cerr << "Arguments missing." << endl;
        cerr << "Usage: VectorOperations <filename>" << endl;

        // return failure, not enough arguments
        return 1;
    }

    // create text input stream connected to file named in argv[1]
    ifstream lInput( argv[1], ifstream::in );

    ...

    return 0;
}
```

Opening a file may fail.

C++ Online References

- https://en.cppreference.com
- https://www.cplusplus.com/reference/



cppreference.com Create account Search

Page Discussion View View source History

C++ reference
C++98, C++03, C++11, C++14, C++17, C++20, C++23

Compiler support (11, 14, 17, 20)
Freestanding implementations
Language
Basic concepts
Keywords
Preprocessor
Expressions
Declaration
Initialization
Functions
Statements
Classes
Overloading
Templates
Exceptions
Headers
Named requirements
Feature test macros (C++20)
Language support library
Type support – traits (C++11)
Program utilities
Coroutine support (C++20)
Three-way comparison (C++20)
numeric_limits – type_info
initializer_list (C++11)

Concepts library (C++20)
Diagnostics library
General utilities library
Smart pointers and allocators
unique_ptr (C++11)
shared_ptr (C++11)
Date and time
Function objects – hash (C++11)
String conversions (C++17)
Utility functions
pair – tuple (C++11)
optional (C++17) – any (C++17)
variant (C++17) – format (C++20)
Strings library
basic_string
basic_string_view (C++17)
Null-terminated strings:
byte – multibyte – wide
Containers library
array (C++11) – vector
map – unordered_map (C++11)
priority_queue – span (C++20)
Other containers:
sequence – associative
unordered associative – adaptors

Iterators library
Ranges library (C++20)
Algorithms library
Numerics library
Common math functions
Mathematical special functions (C++17)
Numeric algorithms
Pseudo-random number generation
Floating-point environment (C++11)
complex – valarray
Localizations library
Input/output library
Stream-based I/O
Synchronized output (C++20)
I/O manipulators
Filesystem library (C++17)
Regular expressions library (C++11)
basic_regex – algorithms
Atomic operations library (C++11)
atomic – atomic_flag
atomic_ref (C++20)
Thread support library (C++11)
thread – mutex
condition_variable

Technical specifications
Standard library extensions (library fundamentals TS)
resource_adaptor — invocation_type
Standard library extensions v2 (library fundamentals TS v2)
propagate_const — ostream_joiner — randint
observer_ptr — detection idiom
Standard library extensions v3 (library fundamentals TS v3)
scope_exit — scope_fail — scope_success — unique_resource
Concurrency library extensions (concurrency TS) — **Transactional Memory** (TM TS)
Concepts (concepts TS) — **Ranges** (ranges TS) — **Reflection** (reflection TS)

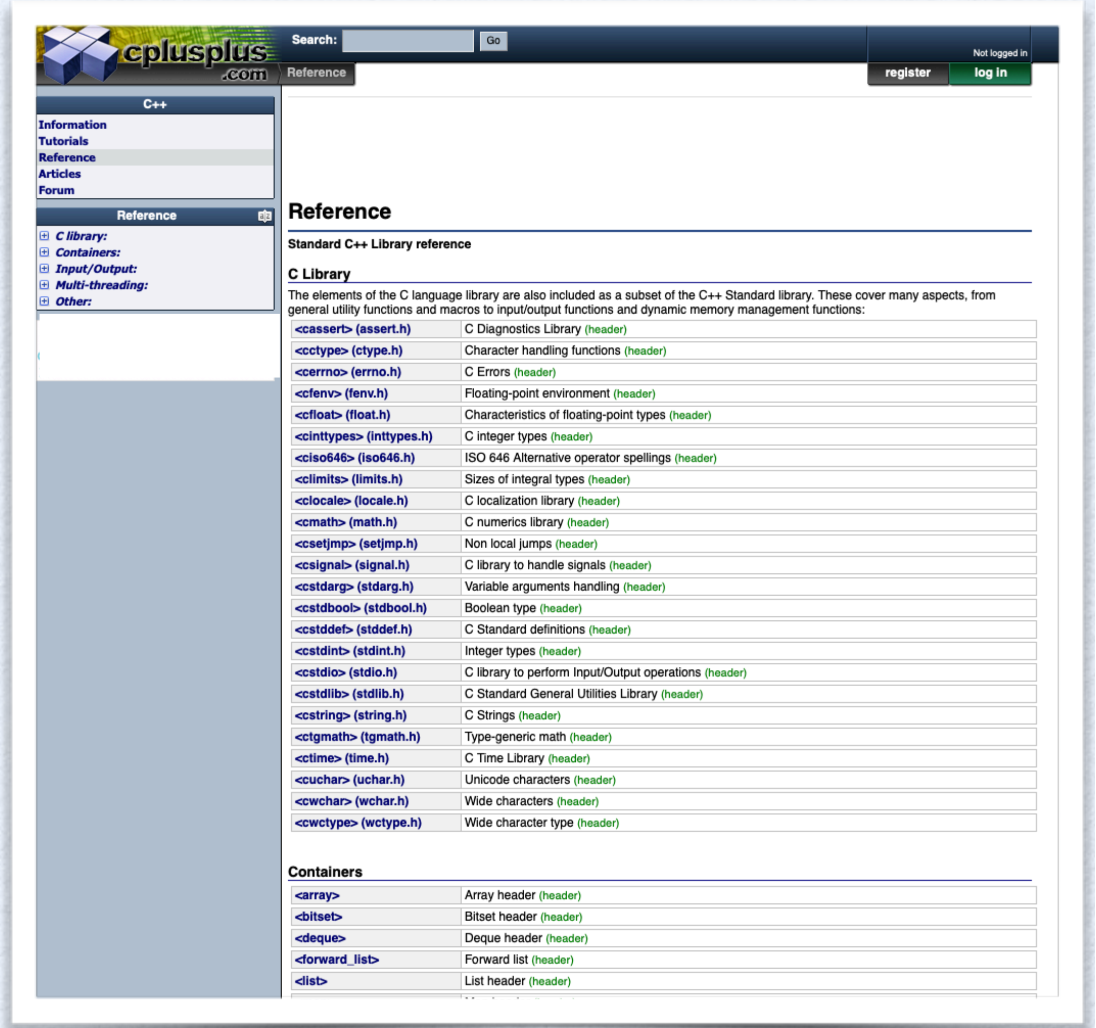
External Links – Non-ANSI/ISO Libraries – Index – std Symbol Index

C reference
C89, C95, C99, C11, C17, C23

Language
Basic concepts
Keywords
Preprocessor
Expressions
Declaration
Initialization
Functions
Statements
Headers

Type support
Program utilities
Variadic functions
Error handling
Dynamic memory management
Date and time utilities
Strings library
Null-terminated strings:
byte – multibyte – wide
Algorithms

Numerics
Common mathematical functions
Floating-point environment (C99)
Pseudo-random number generation
Complex number arithmetic (C99)
Type-generic math (C99)
Input/output support
Localization support
Atomic operations library (C11)
Thread support library (C11)



Reference

Standard C++ Library reference

C Library

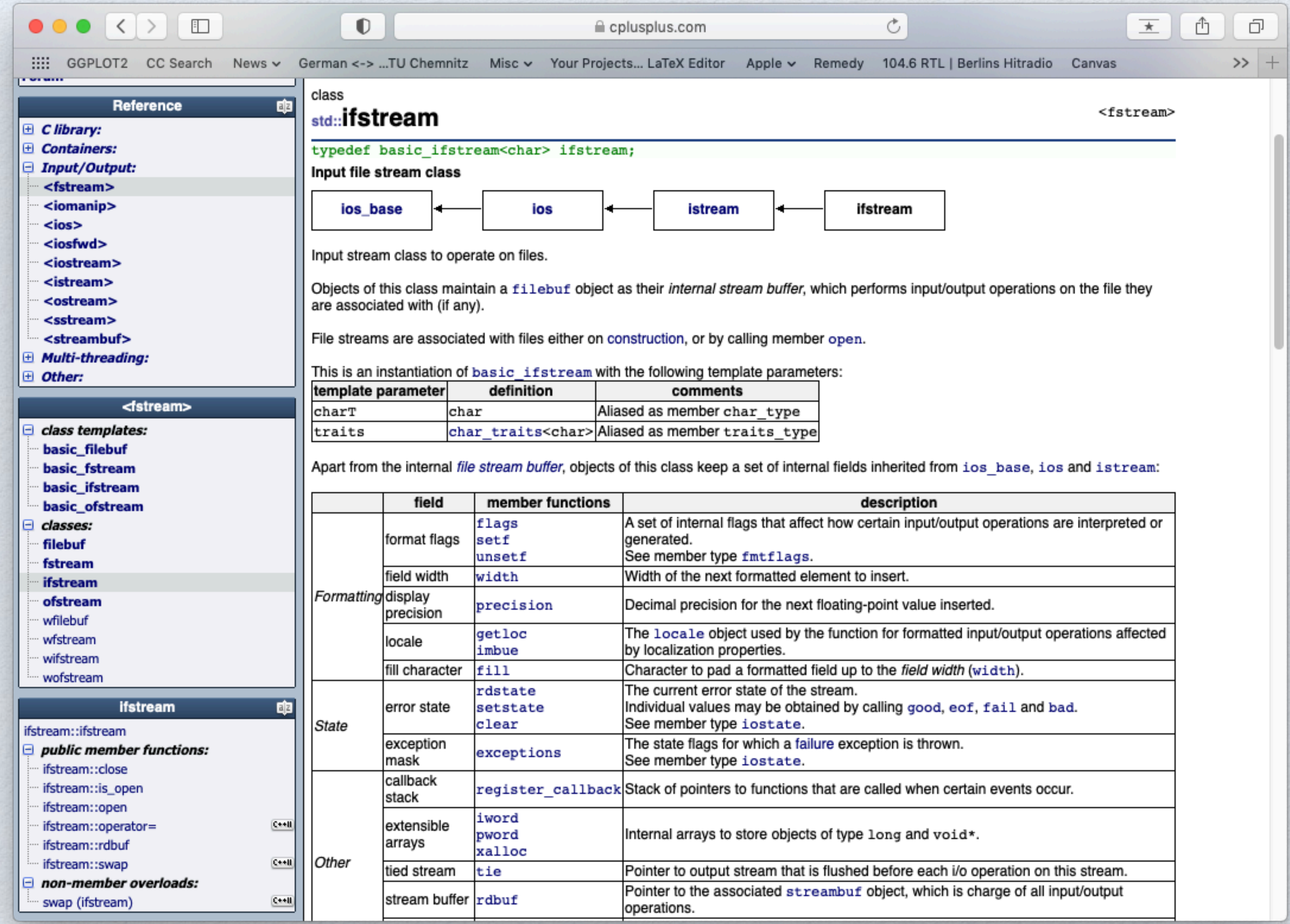
The elements of the C language library are also included as a subset of the C++ Standard library. These cover many aspects, from general utility functions and macros to input/output functions and dynamic memory management functions:

<cassert> (assert.h)	C Diagnostics Library (header)
<cctype> (ctype.h)	Character handling functions (header)
<cerrno> (errno.h)	C Errors (header)
<cfenv> (fenv.h)	Floating-point environment (header)
<cfloat> (float.h)	Characteristics of floating-point types (header)
<cinttypes> (inttypes.h)	C integer types (header)
<ciso646> (iso646.h)	ISO 646 Alternative operator spellings (header)
<climits> (limits.h)	Sizes of integral types (header)
<clocale> (locale.h)	C localization library (header)
<cmath> (math.h)	C numerics library (header)
<csetjmp> (setjmp.h)	Non local jumps (header)
<csignal> (signal.h)	C library to handle signals (header)
<cstdarg> (stdarg.h)	Variable arguments handling (header)
<cstdbool> (stdbool.h)	Boolean type (header)
<cstddef> (stddef.h)	C Standard definitions (header)
<cstdint> (stdint.h)	Integer types (header)
<cstdio> (stdio.h)	C library to perform Input/Output operations (header)
<cstdlib> (stdlib.h)	C Standard General Utilities Library (header)
<cstring> (string.h)	C Strings (header)
<ctgmath> (tgmath.h)	Type-generic math (header)
<ctime> (time.h)	C Time Library (header)
<cuchar> (uchar.h)	Unicode characters (header)
<cwchar> (wchar.h)	Wide characters (header)
<cwctype> (wctype.h)	Wide character type (header)

Containers

<array>	Array header (header)
<bitset>	Bitset header (header)
<deque>	Deque header (header)
<forward_list>	Forward list (header)
<list>	List header (header)

https://www.cplusplus.com/reference



class

std::**ifstream**

`<fstream>`

`typedef basic_ifstream<char> ifstream;`

Input file stream class

$$\begin{tikzcd} \text{ios\_base} & \text{ios} \arrow[l] & \text{istream} \arrow[l] & \text{ifstream} \arrow[l] \end{tikzcd}$$

Input stream class to operate on files.

Objects of this class maintain a `filebuf` object as their *internal stream buffer*, which performs input/output operations on the file they are associated with (if any).

File streams are associated with files either on construction, or by calling member `open`.

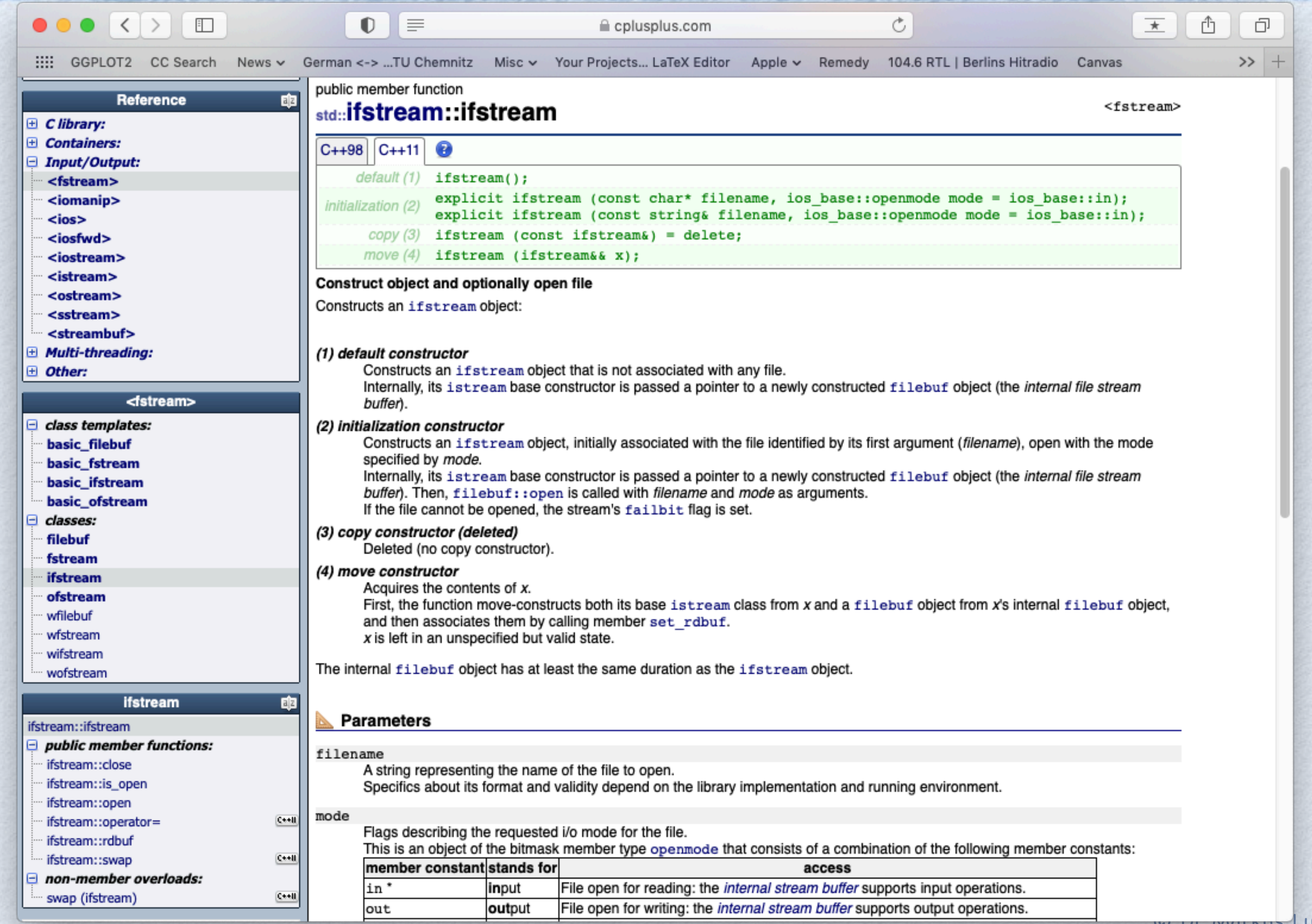
This is an instantiation of `basic_ifstream` with the following template parameters:

template parameter	definition	comments
`charT`	`char`	Aliased as member `char_type`
`traits`	`char_traits<char>`	Aliased as member `traits_type`

Apart from the internal *file stream buffer*, objects of this class keep a set of internal fields inherited from `ios_base`, `ios` and `istream`:

	field	member functions	description
Formatting	format flags	`flags` `setf` `unsetf`	A set of internal flags that affect how certain input/output operations are interpreted or generated. See member type `fmtflags`.
	field width	`width`	Width of the next formatted element to insert.
	display precision	`precision`	Decimal precision for the next floating-point value inserted.
	locale	`getloc` `imbue`	The `locale` object used by the function for formatted input/output operations affected by localization properties.
	fill character	`fill`	Character to pad a formatted field up to the *field width* (`width`).
State	error state	`rdstate` `setstate` `clear`	The current error state of the stream. Individual values may be obtained by calling `good`, `eof`, `fail` and `bad`. See member type `iostate`.
	exception mask	`exceptions`	The state flags for which a failure exception is thrown. See member type `iostate`.
Other	callback stack	`register_callback`	Stack of pointers to functions that are called when certain events occur.
	extensible arrays	`iword` `pword` `xalloc`	Internal arrays to store objects of type `long` and `void*`.
	tied stream	`tie`	Pointer to output stream that is flushed before each i/o operation on this stream.
	stream buffer	`rdbuf`	Pointer to the associated `streambuf` object, which is charge of all input/output operations.

The ifstream Constructor



public member function

std::**ifstream::ifstream**

<fstream>

C++98 C++11

```
   default (1)  ifstream();
initialization (2)  explicit ifstream (const char* filename, ios_base::openmode mode = ios_base::in);
                    explicit ifstream (const string& filename, ios_base::openmode mode = ios_base::in);
      copy (3)  ifstream (const ifstream&) = delete;
      move (4)  ifstream (ifstream&& x);
```

Construct object and optionally open file

Constructs an `ifstream` object:

(1) default constructor
Constructs an `ifstream` object that is not associated with any file.
Internally, its `istream` base constructor is passed a pointer to a newly constructed `filebuf` object (the *internal file stream buffer*).

(2) initialization constructor
Constructs an `ifstream` object, initially associated with the file identified by its first argument (*filename*), open with the mode specified by *mode*.
Internally, its `istream` base constructor is passed a pointer to a newly constructed `filebuf` object (the *internal file stream buffer*). Then, `filebuf::open` is called with *filename* and *mode* as arguments.
If the file cannot be opened, the stream's `failbit` flag is set.

(3) copy constructor (deleted)
Deleted (no copy constructor).

(4) move constructor
Acquires the contents of *x*.
First, the function move-constructs both its base `istream` class from *x* and a `filebuf` object from *x*'s internal `filebuf` object, and then associates them by calling member `set_rdbuf`.
x is left in an unspecified but valid state.

The internal `filebuf` object has at least the same duration as the `ifstream` object.

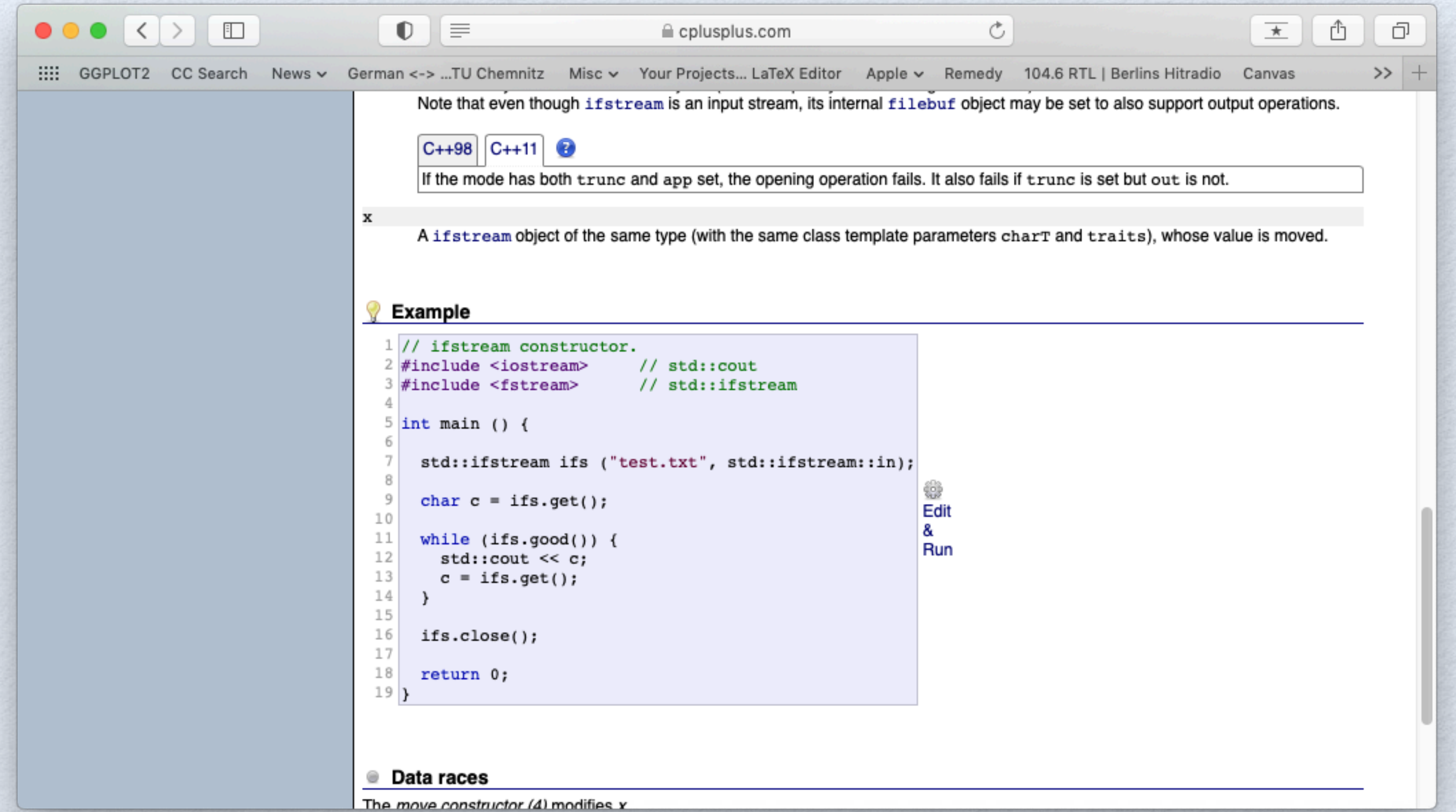
Parameters

`filename`
A string representing the name of the file to open.
Specifics about its format and validity depend on the library implementation and running environment.

`mode`
Flags describing the requested i/o mode for the file.
This is an object of the bitmask member type `openmode` that consists of a combination of the following member constants:

member constant	stands for	access
`in` *	**in**put	File open for reading: the *internal stream buffer* supports input operations.
`out`	**out**put	File open for writing: the *internal stream buffer* supports output operations.

Sample Code


cplusplus.com
GGPLOT2
CC Search
News
German <-> ...TU Chemnitz
Misc
Your Projects... LaTeX Editor
Apple
Remedy
104.6 RTL | Berlins Hitradio
Canvas

Note that even though `ifstream` is an input stream, its internal `filebuf` object may be set to also support output operations.

C++98 C++11

If the mode has both `trunc` and `app` set, the opening operation fails. It also fails if `trunc` is set but `out` is not.

x

A `ifstream` object of the same type (with the same class template parameters `charT` and `traits`), whose value is moved.

Example

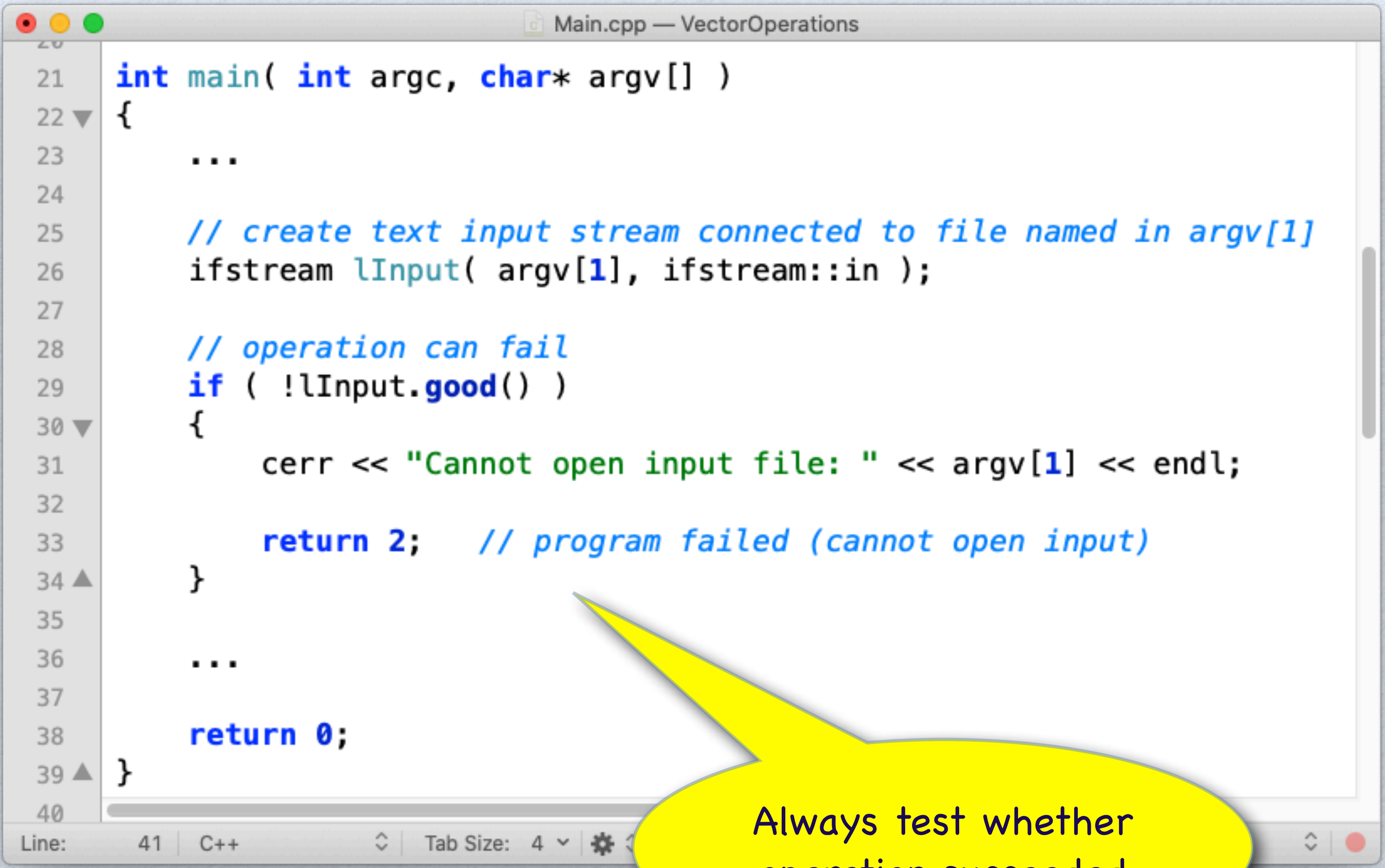
```cpp
// ifstream constructor.
#include <iostream>     // std::cout
#include <fstream>      // std::ifstream

int main () {

  std::ifstream ifs ("test.txt", std::ifstream::in);

  char c = ifs.get();

  while (ifs.good()) {
    std::cout << c;
    c = ifs.get();
  }

  ifs.close();

  return 0;
}
```

Edit & Run

Data races

The *move constructor (4)* modifies *x*.

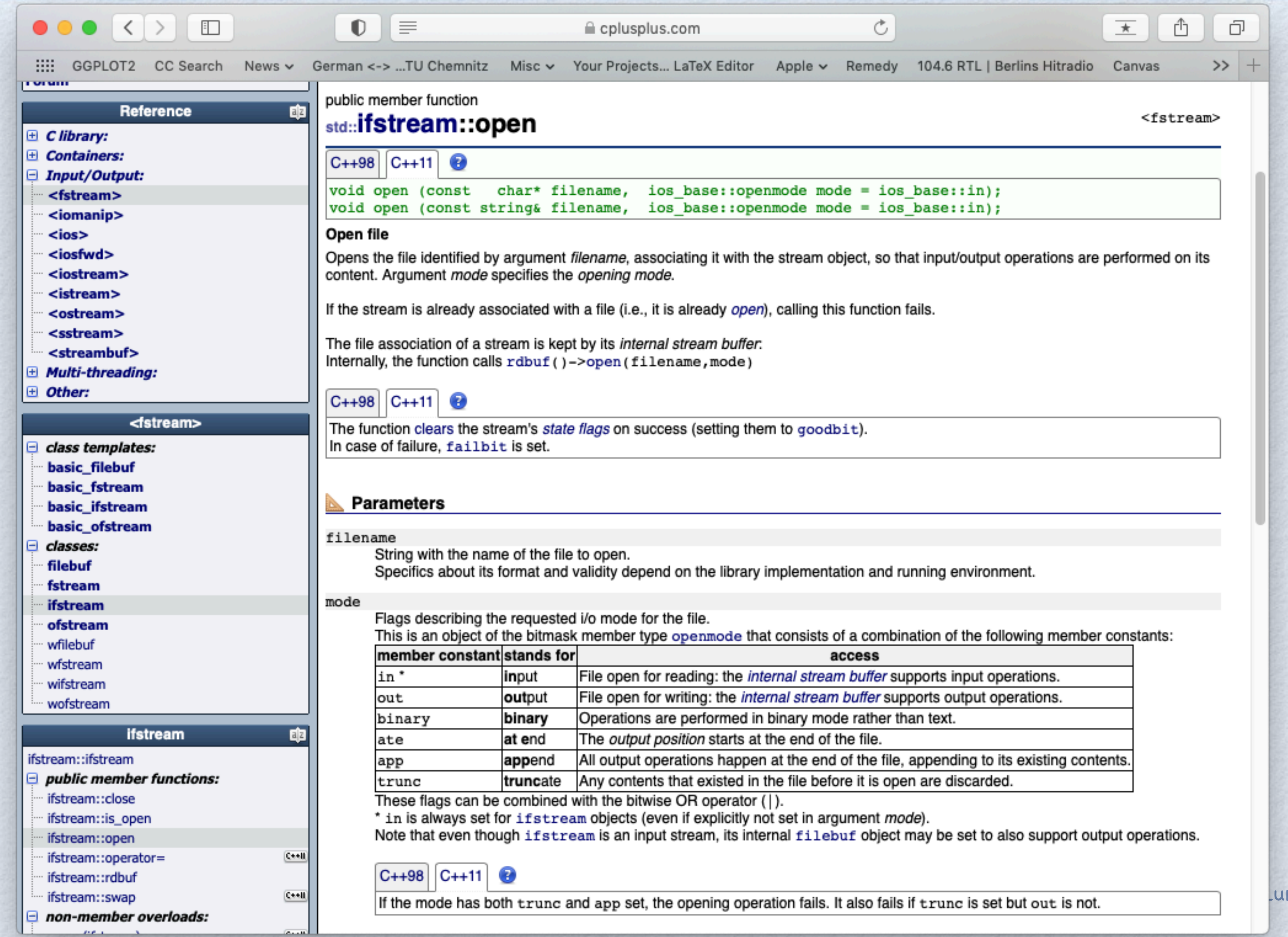
Opening an Input File: Step-by-Step



```cpp
int main( int argc, char* argv[] )
{
    ...

    // create text input stream connected to file named in argv[1]
    ifstream lInput( argv[1], ifstream::in );

    // operation can fail
    if ( !lInput.good() )
    {
        cerr << "Cannot open input file: " << argv[1] << endl;

        return 2;   // program failed (cannot open input)
    }

    ...

    return 0;
}
```

Always test whether operation succeeded

ifstream::open


cplusplus.com
GGPLOT2
CC Search
News
German <-> ...TU Chemnitz
Misc
Your Projects... LaTeX Editor
Apple
Remedy
104.6 RTL | Berlins Hitradio
Canvas

Reference

- C library:
- Containers:
- Input/Output:
  - <fstream>
  - <iomanip>
  - <ios>
  - <iosfwd>
  - <iostream>
  - <istream>
  - <ostream>
  - <sstream>
  - <streambuf>
- Multi-threading:
- Other:

<fstream>

- *class templates:*
  - basic_filebuf
  - basic_fstream
  - basic_ifstream
  - basic_ofstream
- *classes:*
  - filebuf
  - fstream
  - ifstream
  - ofstream
  - wfilebuf
  - wfstream
  - wifstream
  - wofstream

ifstream

- ifstream::ifstream
- *public member functions:*
  - ifstream::close
  - ifstream::is_open
  - ifstream::open
  - ifstream::operator= C++11
  - ifstream::rdbuf
  - ifstream::swap C++11
- *non-member overloads:*

public member function

std::ifstream::open `<fstream>`

C++98 C++11

```
void open (const   char* filename,  ios_base::openmode mode = ios_base::in);
void open (const string& filename,  ios_base::openmode mode = ios_base::in);
```

Open file

Opens the file identified by argument *filename*, associating it with the stream object, so that input/output operations are performed on its content. Argument *mode* specifies the *opening mode*.

If the stream is already associated with a file (i.e., it is already *open*), calling this function fails.

The file association of a stream is kept by its *internal stream buffer*.
Internally, the function calls `rdbuf()->open(filename,mode)`

C++98 C++11

The function clears the stream's *state flags* on success (setting them to `goodbit`).
In case of failure, `failbit` is set.

Parameters

`filename`
String with the name of the file to open.
Specifics about its format and validity depend on the library implementation and running environment.

`mode`
Flags describing the requested i/o mode for the file.
This is an object of the bitmask member type `openmode` that consists of a combination of the following member constants:

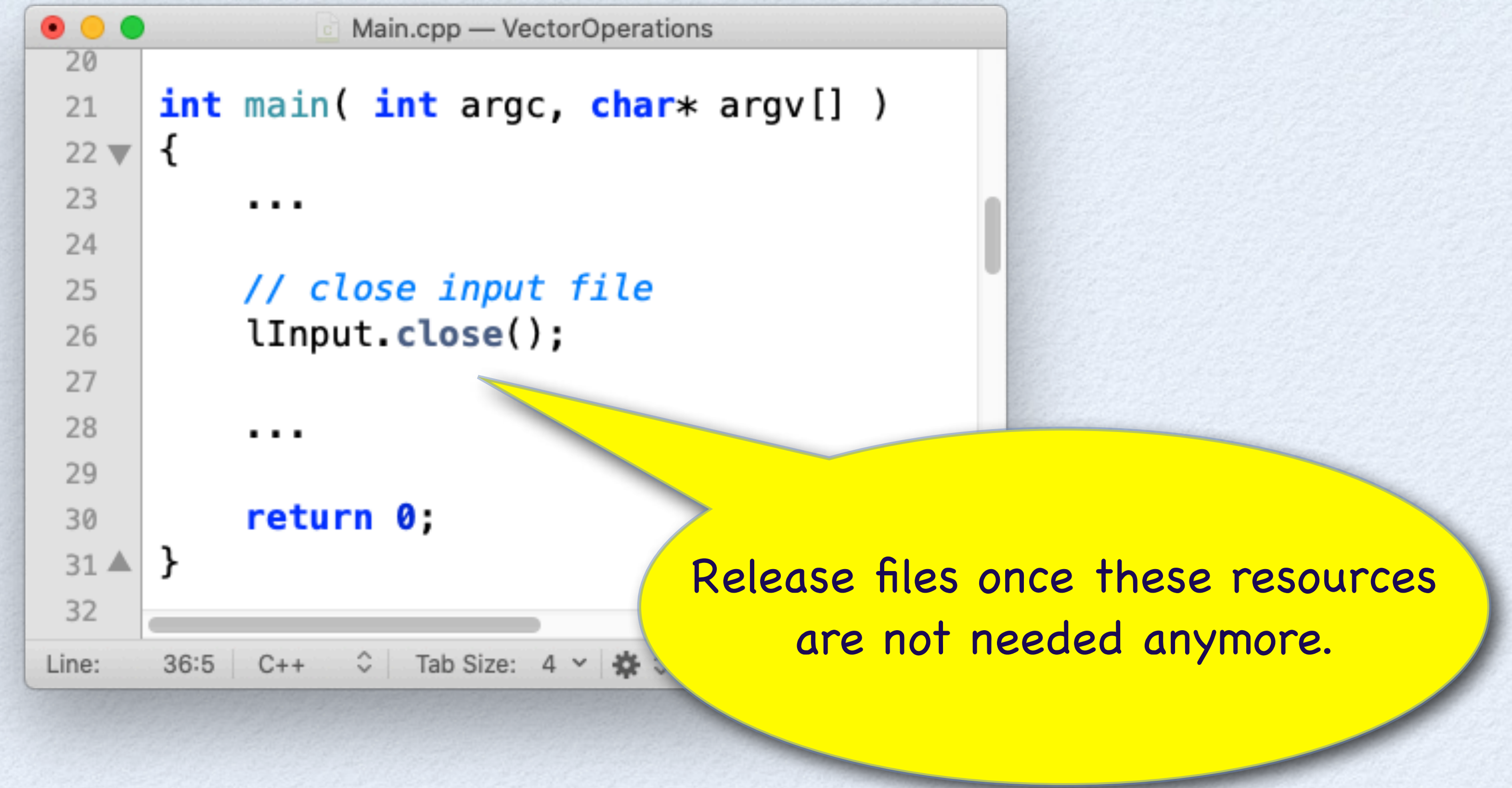
member constant	stands for	access
`in` *	**in**put	File open for reading: the *internal stream buffer* supports input operations.
`out`	**out**put	File open for writing: the *internal stream buffer* supports output operations.
`binary`	**binary**	Operations are performed in binary mode rather than text.
`ate`	**at e**nd	The *output position* starts at the end of the file.
`app`	**app**end	All output operations happen at the end of the file, appending to its existing contents.
`trunc`	**trunc**ate	Any contents that existed in the file before it is open are discarded.

These flags can be combined with the bitwise OR operator (`|`).
* `in` is always set for `ifstream` objects (even if explicitly not set in argument *mode*).
Note that even though `ifstream` is an input stream, its internal `filebuf` object may be set to also support output operations.

C++98 C++11

If the mode has both `trunc` and `app` set, the opening operation fails. It also fails if `trunc` is set but `out` is not.

Closing Files

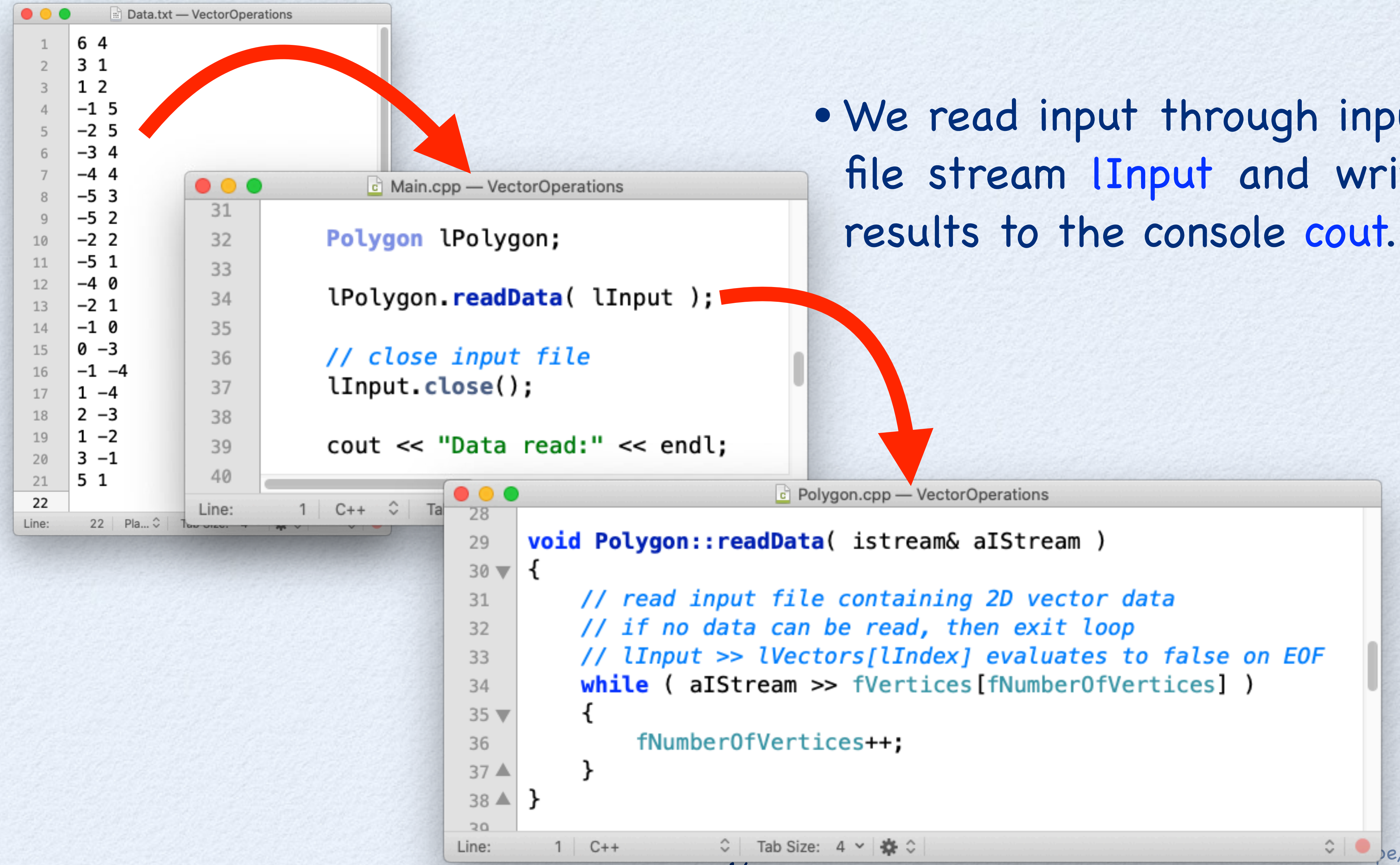


```
int main( int argc, char* argv[] )
{
    ...

    // close input file
    lInput.close();

    ...

    return 0;
}
```

Main.cpp — VectorOperations
Line: 36:5 C++ Tab Size: 4
Release files once these resources are not needed anymore.

- **Note**: Stack-allocated objects are destroyed at the end of their lifetime. The destructor for ifstream calls close. However, your program may not terminate gracefully and hence not call the destructor for your input file objects at all.

Reading Vector2D Data


Data.txt — VectorOperations
Main.cpp — VectorOperations
Polygon.cpp — VectorOperations
Line: 22
Line: 1
C++
Tab Size: 4

```
6 4
3 1
1 2
-1 5
-2 5
-3 4
-4 4
-5 3
-5 2
-2 2
-5 1
-4 0
-2 1
-1 0
0 -3
-1 -4
1 -4
2 -3
1 -2
3 -1
5 1
```

```cpp

    Polygon lPolygon;

    lPolygon.readData( lInput );

    // close input file
    lInput.close();

    cout << "Data read:" << endl;

```

```cpp
void Polygon::readData( istream& aIStream )
{
    // read input file containing 2D vector data
    // if no data can be read, then exit loop
    // lInput >> lVectors[lIndex] evaluates to false on EOF
    while ( aIStream >> fVertices[fNumberOfVertices] )
    {
        fNumberOfVertices++;
    }
}
```

- We read input through input file stream lInput and write results to the console cout.