***What is #include<iostream>?***

* **iostream** is a class in the **std** namespace that abstracts all the behaviors of input and output streams. It is not a “flat” class. It is really the end result of a multiple-inheritance chain that includes the classes **ios** the base class of **istream** and **ostream**, which in turn are the base classes of **iostream**. In addition, it provides implementations for such useful objects as **cout**, **cin**, and **cerr**.
* <iostream> is a header file. This file defines the cin, cout, cerr and clog objects, which corresponds to the standard input stream, the standard output stream, the un-buffered standard error stream, and the buffered standard error stream, respectively.

***#The include<iostream> CODE:***

// Standard iostream objects -\*- C++ -\*-

// Copyright (C) 1997-2013 Free Software Foundation, Inc.

//

// This file is part of the GNU ISO C++ Library. This library is free

// software; you can redistribute it and/or modify it under the

// terms of the GNU General Public License as published by the

// Free Software Foundation; either version 3, or (at your option)

// any later version.

// This library is distributed in the hope that it will be useful,

// but WITHOUT ANY WARRANTY; without even the implied warranty of

// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

// GNU General Public License for more details.

// Under Section 7 of GPL version 3, you are granted additional

// permissions described in the GCC Runtime Library Exception, version

// 3.1, as published by the Free Software Foundation.

// You should have received a copy of the GNU General Public License and

// a copy of the GCC Runtime Library Exception along with this program;

// see the files COPYING3 and COPYING.RUNTIME respectively. If not, see

// <http://www.gnu.org/licenses/>.

/\*\* @file include/iostream

\* This is a Standard C++ Library header.

\*/

//

// ISO C++ 14882: 27.3 Standard iostream objects

//

#ifndef \_GLIBCXX\_IOSTREAM

#define \_GLIBCXX\_IOSTREAM 1

#pragma GCC system\_header

#include <bits/c++config.h>

#include <ostream>

#include <istream>

namespace std \_GLIBCXX\_VISIBILITY(default)

{

\_GLIBCXX\_BEGIN\_NAMESPACE\_VERSION

/\*\*

\* @name Standard Stream Objects

\*

\* The &lt;iostream&gt; header declares the eight <em>standard stream

\* objects</em>. For other declarations, see

\* http://gcc.gnu.org/onlinedocs/libstdc++/manual/bk01pt11ch24.html

\* and the @link iosfwd I/O forward declarations @endlink

\*

\* They are required by default to cooperate with the global C

\* library's @c FILE streams, and to be available during program

\* startup and termination. For more information, see the HOWTO

\* linked to above.

\*/

//@{

extern istream cin; /// Linked to standard input

extern ostream cout; /// Linked to standard output

extern ostream cerr; /// Linked to standard error (unbuffered)

extern ostream clog; /// Linked to standard error (buffered)

#ifdef \_GLIBCXX\_USE\_WCHAR\_T

extern wistream wcin; /// Linked to standard input

extern wostream wcout; /// Linked to standard output

extern wostream wcerr; /// Linked to standard error (unbuffered)

extern wostream wclog; /// Linked to standard error (buffered)

#endif

//@}

// For construction of filebuffers for cout, cin, cerr, clog et. al.

static ios\_base::Init \_\_ioinit;

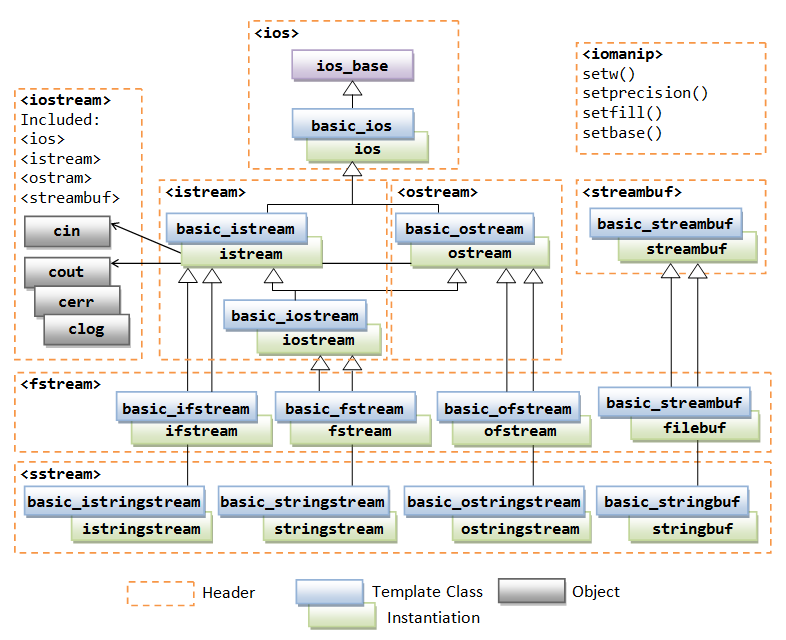
\_GLIBCXX\_END\_NAMESPACE\_VERSION

} // namespace

#endif /\* \_GLIBCXX\_IOSTREAM \*/

To fully understand **iostream**, in addition to the object-oriented paradigm the programmer has to be familiar with the concept of streams, multiple inheritance and directed acyclic graphs (DAGs)—Understanding DAGs is needed to see why and where virtual inheritance is required to avoid violating the ***one-definition rule—***. In short, to be proficient a C++ programmer must learn and study the C++ language very well. The resulting superior knowledge of such programmer significantly increases the quality of the code (s)he produces.

The C/C++ languages require skills not needed in other computer languages. Therefore, I advise prospective users of the language that one **should not** claim expertise in C/C++ programming simply because one refactored into C/C++ programs written in other languages.



**“What does namespace means in c++ program?”**

A **namespace** is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries. All identifiers at namespace scope are visible to one another without qualification. Identifiers outside the namespace can access the members by using the fully qualified name for each identifier, for example std::vector<std::string> vec;, or else by a [using Declaration](https://docs.microsoft.com/en-us/cpp/cpp/using-declaration?view=vs-2017) for a single identifier (using std::string), or a [using Directive](https://docs.microsoft.com/en-us/cpp/cpp/namespaces-cpp?view=vs-2017#using_directives) for all the identifiers in the namespace (using namespace std;). Code in header files should always use the

**What does ‘int(main)’ means?**

The purpose of **main** 's return value is to return an exit status to the operating system. In standard C, the only valid signatures for **main** are: **int main**(void) and **int main**(**int** argc, char \*\*argv) The form you're using: **int main()** is an old style declaration that indicates **main** takes an unspecified number of arguments.

**-*SimpleyVoy07***