头文件

在 C 语言家族程序中，头文件被大量使用。一般而言，每个 C++/C 程序通常由头文件和定义文件组成。头文件作为一种包含功能函数、数据接口声明的载体文件，主要用于保存程序的声明，而定义文件用于保存程序的实现。

历史

C++ 是在 C 语言的基础上开发的，早期的 C++ 还不完善，不支持命名空间，没有自己的编译器，而是将 C++ 代码翻译成 C 代码，再通过 C 编译器完成编译。这个时候的 C++ 仍然在使用 C 语言的库，stdio.h、stdlib.h、string.h 等头文件依然有效；此外 C++ 也开发了一些新的库，增加了自己的头文件，例如：

iostream.h：用于控制台输入输出头文件。

fstream.h：用于文件操作的头文件。

complex.h：用于复数计算的头文件。

和 C 语言一样，C++ 头文件仍然以.h 为后缀，它们所包含的类、函数、宏等都是全局范围的。

后来 C++ 引入了命名空间的概念，计划重新编写库，将类、函数、宏等都统一纳入一个命名空间，这个命名空间的名字就是 std。std 是 standard 的缩写，意思是“标准命名空间”。

但是这时已经有很多用老式 C++ 开发的程序了，它们的代码中并没有使用命名空间，直接修改原来的库会带来一个很严重的后果：程序员会因为不愿花费大量时间修改老式代码而极力反抗，拒绝使用新标准的 C++ 代码。

C++ 开发人员想了一个好办法，保留原来的库和头文件，它们在 C++ 中可以继续使用，然后再把原来的库复制一份，在此基础上稍加修改，把类、函数、宏等纳入命名空间 std 下，就成了新版 C++ 标准库。这样共存在了两份功能相似的库，使用了老式 C++ 的程序可以继续使用原来的库，新开发的程序可以使用新版的 C++ 库。

为了避免头文件重名，新版 C++ 库也对头文件的命名做了调整，去掉了后缀.h，所以老式 C++ 的 iostream.h 变成了 iostream，fstream.h 变成了 fstream。而对于原来 C 语言的头文件，也采用同样的方法，但在每个名字前还要添加一个 c 字母，所以 C 语言的 stdio.h 变成了 cstdio，stdlib.h 变成了 cstdlib。

需要注意的是，旧的 C++ 头文件是官方所反对使用的，已明确提出不再支持，但旧的 C 头文件仍然可以使用，以保持对 C 的兼容性。实际上，编译器开发商不会停止对客户现有软件提供支持，可以预计，旧的 C++ 头文件在未来数年内还是会被支持。

分类

传统 C++

```
#include<assert.h>//设定插入点
#include<ctype.h>//字符处理
#include<errno.h>//定义错误码
#include<float.h>//浮点数处理
#include<fstream.h>//文件输入/输出
#include<iomanip.h>//参数化输入/输出
#include<iostream.h>//数据流输入/输出
#include<limits.h>//定义各种数据类型最值常量
```

```
#include<locale.h>//定义本地化函数
#include<math.h>//定义数学函数
#include<stdio.h>//定义输入/输出函数
#include<stdlib.h>//定义杂项函数及内存分配函数
#include<string.h>//字符串处理
#include<strstrea.h>//基于数组的输入/输出
#include<time.h>//定义关于时间的函数
#include<wchar.h>//宽字符处理及输入/输出
#include<wctype.h>//宽字符分类
```

标准 C++

```
#include<algorithm>//STL 通用算法
#include<bitset>//STL 位集容器
#include<bits/stdc++.h>//编译器 GCC 4.8 支持的万能头文件，基本包含所有头文件
#include<cctype>//C 字符处理
#include<cerrno>//C 的错误报告机制
#include<clocale>
#include<cmath>//兼容 C 语言数学库
#include<complex>//复数类
#include<cstdio>//C 语言输入输出工具
#include<cstdlib>//C 语言通用工具
#include<cstring>//C 字符串
#include<ctime>
#include<deque>//STL 双端队列容器
#include<exception>//异常处理类
#include<fstream>//文件输入输出流
#include<functional>//STL 定义运算函数（代替运算符）
#include<limits>
#include<list>//STL 线性列表容器
#include<map>//STL 映射容器
#include<iomanip>
#include<ios>//基本输入/输出支持
#include<iosfwd>//输入/输出系统使用的前置声明
#include<iostream>//基本输入输出流
#include<queue>//STL 队列容器
#include<set>//STL 集合容器
#include<sstream>//基于字符串的流
#include<stack>//STL 堆栈容器
#include<stdexcept>//标准异常类
#include<streambuf>//底层输入/输出支持
#include<string>//字符串类
#include<utility>//STL 通用模板类
#include<vector>//STL 动态数组容器
#include<cwchar>
#include<cwctype>
```

在 C++中，标准库的命名空间为 std，因而包含了上述头文件时，一般会使用下列语句：

```
using namespace std;
```

C99 版本

```
#include<complex.h>//复数处理
#include<fenv.h>//浮点环境
#include<inttypes.h>//整数格式转换
#include<stdbool.h>//布尔环境
#include<stdint.h>//整型环境
#include<tgmath.h>//通用类型数学宏
```

引用头文件

只引用一次头文件

如果一个头文件被引用两次，编译器会处理两次头文件的内容，这将产生错误。为了防止这种情况，标准的做法是把文件的整个内容放在条件编译语句中，如下：

```
#ifndef HEADER_FILE
#define HEADER_FILE

the entire header file file

#endif
```

这种结构就是通常所说的包装器 #ifndef。当再次引用头文件时，条件为假，因为 HEADER_FILE 已定义。此时，预处理器会跳过文件的整个内容，编译器会忽略它。

有条件引用

有时需要从多个不同的头文件中选择一个引用到程序中。例如，需要指定在不同的操作系统上使用的配置参数。您可以通过一系列条件来实现这点，如下：

```
#if SYSTEM_1
   # include "system_1.h"
#elif SYSTEM_2
   # include "system_2.h"
#elif SYSTEM_3
   ...
#endif
```

但是如果头文件比较多的时候，这么做是很不妥当的，预处理器使用宏来定义头文件的名称。这就是所谓的有条件引用。它不是用头文件的名称作为 #include 的直接参数，您只需要使用宏名称代替即可：

```
#define SYSTEM_H "system_1.h"
...
#include SYSTEM_H
```

SYSTEM_H 会扩展，预处理器会查找 system_1.h，就像 #include 最初编写的那样。SYSTEM_H 可通过 -D 选项被您的 Makefile 定义。

统一头文件管理

在有多个 .h 文件和多个 .c 文件的时候，往往我们会用一个 global.h 的头文件来包括所有的 .h 文件，然后在除 global.h 文件外的头文件中 包含 global.h 就可以实现所有头文件的包含，同时不会乱。方便在各个文件里面调用其他文件的函数或者变量。

```
#ifndef _GLOBAL_H
#define _GLOBAL_H
#include <fstream>
#include <iostream>
#include <math.h>
#include <Config.h>
```

include <> 与 include "" 的区别

#include < > 引用的是编译器的类库路径里面的头文件。

#include " " 引用的是你程序目录的相对路径中的头文件，如果在程序目录没有找到引用的头文件则到编译器的类库路径的目录下找该头文件。