PCLint选项详解

目 录

以下为PC-lint for C/C++ (NT) Ver. 7.50v版本配置参数的详细解释及用法举例。

LINT选项可以放在注释中，例如：

/*lint option1 option2 ... optional commentary */　选项可以有多行

//lint option1 option2 ... optional commentary　　选项仅为一行

选项间要以空格分开，lint命令一定要小写，并且紧跟在/*或//后面，不能有空格。如果选项由类似于操作符和操作数的部分组成，例如-esym(534, printf, scanf, operator new)，其中最后一个选项是operator new，那么在operator和new中间只能有一个空格。

选项还可以放在宏定义中，当宏被展开时选项才生效。例如：

#define DIVZERO(x) /*lint -save -e54 */ ((x) /0) /*lint -restore */　允许除数为0而不告警

LINT的选项很多共有300多种，大体可分为以下几类：

1 错误信息禁止选项

说明：　“-”　：　表示禁止输出相应的错误消息

“+”　：　表示允许输出相应的错误消息

“#”　：　允许使用通配符“?”和“*”

除了900级别（900－999）和1900（1900－1999）级别的告警消息缺省是关闭的外，其它的告警消息缺省均是打开的。

-e#　：　禁止输出告警号为#的消息

-e(#)　：　对于下一个表达式禁止输出告警号为#的消息

!e#　：　在本行禁止输出告警号为#的消息

--e(#)　：　对当前的整个表达式禁止输出告警号为#的消息

-eai　：　整型数子类参数不一致，如：char/short vs. int

-ean　：　名义上的参数不一致，如：字节数相同（都是32位）的int和long等

-eas　：　参数大小相同，如：如果int和pointer字节数相同，那么如果f()的参数应该是pointer的话，用f(3)整型数调用就会报错，设置此项可以关闭告警

-eau　：　参数类型一致，但是符号类型不一致，如：unsigned int和int

以上四个选项主要用于非原型的旧风格的C语言程序。其中eas涵盖了ean和eau。

-efile(#,<File>)　对指定文件禁止输出告警号为#的消息

-efunc(#,<Func>)　对于函数Func，禁止输出告警号为#的消息

-elib(#)　：　对于库头文件禁止输出告警号为#的消息

-elibsym(#)　：　对于所有库头文件中的符号禁止输出告警号为#的消息，此告警不同于elib之处在于-elib(#)仅仅当分析头文件时不输出相应的告警，如果你在源程序中使用了会导致告警#的变量等，在分析源程序时还是会告警的，因此要想完全的关闭该告警，使其在头文件和源文件中均不出现，请使用本选项

-emacro(#,Symbol) 对于宏Symbol，当其展开时禁止输出告警号为#的消息

-emacro((#),Symbol)对于宏Symbol，当其展开时禁止输出告警号为#的消息，与上一个选项的区别是它会先将宏加上一对括号再判断，如：#define DIVIDE(n , m) n / m 那么它会在宏展开时将n / m看作(n / m)来处理。用处不是很大。

-epn ： 名义上的指针不一致，如：对于指向字节数大小相同的变量的指针

-eps ： 指针指向的类型不同，但大小字节数相同

-epu ： 指针指向的类型仅仅符号不一致

-epp ： 指针指向的类型不确定

-epuc ： 指针指向的字符串类型，其符号不一致

-epnc ： 指针指向的字符串类型，仅仅名义上不同

-esym(#,Symbol) 对于指定的符号Symbol（可以是变量名、函数名等），禁止输出告警号为#的消息，符号Symbol中可以使用通配符*和?。-e#的级别比较高，因此对于-e714 +esym(714,alpha)，后一个选项将不起作用

-etd(<TypeDiff>) 对于TypeDiff类型，忽略不同地方对其的类型定义不同，用于旧风格C

-w<Lev> ： 设置告警级别(0,1,2,3,4)，0表示不打印任何告警消息，用于先关闭所有告警，然后打开部分告警

-wlib(<Lev>) 对库（文件）设置告警级别

2 变量类型大小选项

说明：不同的目标机、编译系统变量类型的的大小（如短整型变量、整型变量等）会有所不同，该类选项用于为目标机设置变量类型的大小。

-sb# ： 设置一个字节的比特数，缺省值为8

-sc# ： sizeof(char)，缺省值为1

-slc# ： sizeof(long char)，缺省值为2

-ss# ： sizeof(short)，缺省值为2

-si# ： sizeof(int)，缺省值为4

-sl# ： sizeof(long)，缺省值为4

-sll# ： sizeof(long long)，缺省值为8

-sf# ： sizeof(float)，缺省值为4

-sd# ： sizeof(double)，缺省值为8

-sld# ： sizeof(long double)，缺省值为16

-sp# ： sizeof(all pointers)，缺省值为4和6

-spN# ： size of near ptrs，缺省值为4

-spF# ： size of far ptrs ，缺省值为6

-spND# ： size of near data pointer，缺省值为4

-spNP# ： size of near prog pointer，缺省值为4

-spFD# ： size of far data pointer，缺省值为6

-spFP# ： size of far prog pointer，缺省值为6

-spD# ： size of data ptrs，缺省值为4和6

-spP# ： size of program ptrs，缺省值为4和6，near为4，far为6

-smp# ： size of all member ptrs，缺省值为4

-smpD# ： size of member ptr (data)，缺省值为4

-smpP# ： size of member ptrs (prog)，缺省值为4

-smpNP# ： size of member ptr (Near Prog)，缺省值为4

-smpFP# ： size of member ptr (Far Prog)，缺省值为4

-sw# ： size of wide char，缺省值为2

3 冗余信息选项

说明： “-” ： 表示仅输出到文件stdout

“+” ： 表示同时输出到文件stdout和stderr

使用格式 ： {-+}v[oish#][mf<int>]

冗余信息指的是LINT过程中产生的一些与编译过程有关的信息，而不是指真正的告警信息、错误信息等。是否生成这些信息可以通过-v和+v选项来决定。+v是生成这些信息，-v是关闭这些信息，这组选项中除+v外，其它所有选项都可以关闭+v选项。

以下选项可以出现0或多个：

o ： 输出命令行和注释行中包含的配置

i ： 输出所使用的配置文件名（lnt文件）

s ： 输出内存消耗数量

h ： 输出严格的类型层次图（输出结果类似于DOS命令tree的结果）

： 附加文件ID，用于判断文件是否相同

以下选项一次只能出现一个：

m ： 输出模块名（缺省就是输出模块名）

f ： 输出所有文件名（此选项包含m选项）

<int> ： 每条消息打印为int行（此选项包含f选项）

4 标志选项

说明： “+” ： 表示打开该开关（即设置标志为1）

“-” ： 表示关闭该开关（即设置标志为0）

“++” ： 表示增1

“--” ： 表示减1

标志选项用于打开或关闭对某类语法情况的处理。后面两个选项（++和--）用于你想在局部改变开关取值，而不影响全局设置的情况。

例如你可以这样使用：

```
/*lint ++flb */
int printf( );
/*lint --flb */
```

fab ： 支持缩略式结构，如：s.a.b如果不会引起歧义的话，可以缩写为s.b。很少有编译器支持这种功能。缺省值为OFF

fai ： 指针参数是否初始化（不初始化将告警），缺省值为ON

fan ： 是否支持匿名联合，用于结构变量的说明，对C缺省值为OFF，C++为ON

fas ： 是否支持匿名结构，类似于fan，缺省值同上

fba ： Bit位寻址能力（可访问性），缺省值为OFF

fbc ： 是否支持布尔类型常量0b...，缺省值为OFF

fbo ： 是否允许bool, true, false关键字，缺省值为ON

fbu ： 是否强制比特位域为无符号数，缺省值为OFF

fcd ： 是否区分cdecl说明符，缺省值为OFF

fce ： 遇到#error时是否继续运行，缺省值为OFF

fcp ： 强制使用C++处理方式，缺省值为OFF

+fcp表示其后的模块文件将被强制作为C++程序对待处理。-fcp表示按照扩展名使用缺省的处理方式。例如：lin +fcp a1.c a2.c a3.c -fcp a4.c a5.cpp，其中a1.c、a2.c和a3.c将被作为C++程序进行Lint，而a4.c和a5.cpp按照缺省的方式（C和C++）进行Lint。

fct ： 生成标签（说明：结构名、枚举名就是一种标签 Tag），缺省值为OFF

fcu ： char型总是作为unsigned，缺省值为OFF

fdc ： C++中是否区分char、unsigned char、signed char，缺省值为ON

fdh ： 是否为头文件名附加'.h'，缺省值为OFF

fdi ： 是否从头文件目录（而非当前目录）开始搜索头文件，缺省值为OFF

fdl ： 是否指针之差为long型数，缺省值为OFF，即int型数

fdr ： 检查函数的返回模式（返回值是否被使用等，旧风格的C），缺省值为OFF

feb ： 是否允许枚举类型做位域，缺省值为ON

fem ： 是否支持非紧靠式修饰符，如：pascal int f();和int pascal f();对于某些编译器只支持后一种紧靠式说明，缺省值为OFF

ffb ： For语句生成代码块（Block），缺省值为ON

ffd ： 是否使float拓宽为等于double，缺省值为OFF

ffn ： 输出文件的全路径名，缺省值为OFF

ffo ： 在生成每条告警后是否调用fflush()立即存入文件，缺省值为ON

fhd ： 是否采用增强型类型定义层次判断，缺省值为ON

fhg ： 是使用图形字符（ON）还是ASCII字符（OFF）来显示类型层次图，缺省值为ON

fhs ： 是否自动将基于typedef的强类型形成类型层次，缺省值为ON

fhx ： 强索引类型是否通过类型层次相关，缺省值为ON

fie ： 等同枚举类型和整型，缺省值为OFF

fil ： 是否对标号进行缩排检查，缺省值为OFF

fim ： 能否使用-i包含多个包含路径，缺省值为ON

fiq ： 忽略缺省的限定词，如Large模式下的far就是可忽略的，缺省值为OFF

fis ： 整型常量是否有符号，如对常量0xFFFF，此标志为On时解释为负数，Off时解释为正数，缺省值为OFF

fkp ： 是否使用K&R预处理方式（非ANSI的C标准），缺省值为OFF

flb ： 是否强制将其作为库程序代码对待，缺省值为OFF

flc ： 是否允许使用long char类型变量，缺省值为OFF

flf ： 是否分析库函数定义(C++)，缺省值为OFF

fll ： 是否允许使用long long int类型变量，缺省值为OFF

fln ： 是否不忽略#line命令，缺省值为ON

fmd ： 是否允许多次重复定义，缺省值为OFF

fna ： (C++)是否允许使用操作符new[]，缺省值为ON

fnc ： 是否允许使用嵌套注释，缺省值为OFF

fnt ： (C++)class类中是否允许嵌套使用struct,union等，缺省值为ON

fod ： 生成Lob文件时是否输出所有对象声明用于模块间检查，缺省值为OFF

fol ： 生成Lob文件时是否输出所有库中对象，缺省值为OFF

fpa ： 退出前是否暂停，缺省值为OFF

fpc ： 指针强制类型转换后是否保留左值特性，缺省值为OFF

例如：使int指针pi指向下一个字节的地址，标准用法是(* (char **) &pi) ++;一种普遍的简化方法是((char *) pi) ++;但是后面一种用法对于标准C来说强制类型转化使pi失去了左值特性，因此无法对其进行++操作，如果你的编译器支持后一种写法，请将本开关置为ON。

fpm ： 是否抑止“缺失精度”告警，缺省值为OFF

fpn ： 是否假定所有指针均有可能为NULL，因此如果使用指针前不判断是否为NULL的话就会告警，缺省值为OFF

fps ： 宏参数是否能在字符串内进行替换，用于非ANSI程序，对于ANSI标准有#连字符可以完成同样功能，缺省值为OFF

frb ： 是否自动为fopen调用增加参数"rb"，缺省值为OFF

fsa ： 是否允许结构赋值，缺省值为ON

fsc ： 是否将字符串作为const char *类型，缺省值为OFF

fsh ： 是否使用共享式读文件打开，缺省值为OFF

fsu ： 是否将字符串作为unsigned类型，缺省值为OFF

ftf ： 是否处理未加工的模板函数，缺省值为OFF

ftr ： 是否将文件名截断为8.3格式，缺省值为OFF

ful ： 是否支持unsigned long类型，缺省值为ON

fva ： 是否将函数当作可变长参数函数处理，缺省值为OFF，

此参数为ON时表示不对该函数进行参数检查，这对于printf()，fprintf()等函数可以抑止参数告警（如515、516告警）

fvo ： 是否支持void类型，缺省值为ON

fvr ： 返回方式是否可变（如不使用返回值等），缺省值为OFF

fwc ： 是否将wchar_t作为标准类型，缺省值为OFF

fwu ： 将wchar_t当作unsigned类型，缺省值为OFF，此选项设为ON将导致+fwc

fxa ： 严格的数组类型检查，即对于数组参数只能传类型相同的数组实参，缺省值为OFF

fxc ： 严格的字符类型检查，即对于char不能转化为int去匹配，缺省值为OFF

fxf ： 严格的float类型检查，即对于float不能转化为double去匹配，缺省值为OFF

fxs ： 严格的short类型检查，即对于short不能转化为int去匹配，缺省值为OFF

以上四个选项只用于非原型类的旧风格函数。

fzl ： 定义sizeof()返回值为long，缺省值为OFF

fzu ： 定义sizeof()返回值为unsigned，缺省值为ON

5 输出格式选项

1.控制消息高度，使用格式为：-h[abfFrsm/<M>/]<I>]<ht>（缺省值为 -ha_3）

<ht> ： 为消息高度，取值范围为1－4

<I> ： 为每条消息的位置指示符，如果<ht>为2,该指示符将与消息同行显示，如果<ht>大于2，该符号将单独在一行显示

a ： 表示将位置指示符放在源程序中错误行的前一行（<ht>必须为3或4才行）

b ： 表示将位置指示符放在源程序中错误行的下一行（<ht>必须为3或4才行）

f ： 消息中总是包含文件信息

F ： 消息中总是包含文件信息，对于无法定位的行号使用MaxLine＋1

r ： 对于针对源文件中同一行的多个错误消息，每个都重复显示行号

s ： 消息间空一行显示

m/<M>/ ： 恢复对宏定义的显示，<M>为宏定义的指示符，缺省为“#”

mn ： 对于源文件中由宏使用产生的错误，不显示该宏的定义

2.控制消息宽度，使用格式为：-width(<Width>,<Indent>)（缺省值为 -width(79,4)）

3.控制消息格式，使用格式为：-format=...（对于消息高度为4的情况，此选项无用。应该使用-format4a和-format4b）

%f ： 文件名

%m ： 告警消息文本

%n ： 告警号

%t ： 消息类型（Error，Warning等）

%l ： 行号

%c ： 列号

%C ： 列号+1

%(...%) ： 如果错误发生在一个文件内，那么包含...信息

\n ： 回车

\t ： Tab键

\s ： 空格键

\a ： 告警（ASCII 7）

\q ： 引号（“ ”）

\\ ： 反斜线（' \ '）

6 其它选项

-A ： 严格使用ANSI C/C++处理方式

-/+b ： 抑止/重定向标语行（包括PCLint的版本号和Copyright信息）到stdout，此选项必须放在命令行而不能放在配置文件中

-c<code> ： 指定一个特定的编译器，如果你使用一个编译器的配置文件如co-code.lnt，那么与该编译器相关的-c<code>选项会自动设置上

-/+cpp(extension) 减少/增加以 .ext 作为扩展名的文件作为C++文件进行Lint处理。例如：+cpp(cc)表示以cc为扩展名的文件（如a.cc）将被作为C++文件对待

-d<name>[=<value>]

定义预处理用的宏，这些宏将在当前模块及其后续模块中有效。如果没有=value，将缺省为其赋值为1，如果只是没有value而有=的话，将缺省赋值为空。例如：

-dDOS

-dalpha=0

-dX=

等同于如下定义：

```
#define    DOS    1
#define    alpha  0
#define    X
```

-D<nm>[=<val>][;<nm>[=<val>]]...

定义宏标号集合，此选项一次可以定义多个宏，其余同上。使用+d... 或 +D...是同样的意思，而且定义更精确。（原文：except it locks in definitions）

+ext(ext[,ext]...)

指定PCLint**能够处理的文件扩展名及处理顺序。缺省值为** +ext(lnt,cpp,c)**。就是说对于没有带扩展名的文件将按照这个顺序进行查找匹配。如：**lint alpha**，将顺序查找**alpha.lnt、alpha.cpp、alpha.c**。注意**Unix**下，扩展名是区分大小写的。**

-father(Parent,Children)

-parent**选项的更严格的形式。即对于强类型，**Child**类型可以赋值给**Parent**类型，但是反过来不行。其它的作用与**-parent**参数一样。**

-function(f0,f1[,f2]...)

使函数f1 (, f2 ...)**象**f0**一样。由于**PCLint**对于一些系统调用的函数有特殊处理，如对于**fopen**函数会检查其两个参数是否为**NULL**。而这个选项的目的是让你的函数象这些系统函数一样，接受这些特殊的检查。具体可参看随盘的手册。**

-header(file)　　**强制**PCLint**在每个模块的开始读入头文件**file

-i<directory>　　**指定包含文件的路径，**-i-**用于取消以前用**-i**建立的所有包含路径**

-ident(<chars>)　　**增加可以作为标志符的字符**

-idlen(<n>[,opt])

报告两个标志符前n**个字符相同，其它不同的情况。一般对于**linker**程序，预处理程序和编译器程序，能识别的标志符的长度都是有限的。因此用该选项可以找出那些在可识别的范围内（如标志符最长只能为**8**个字符）其实是同名的标志符。**opt**可以取值为**x、p**和**c**。x**表示**external**，用于**linker**时模块间的符号；p**表示**preprocessor**，用于预处理时使用的符号；c**表示**compiler**，用于编译时的符号。如果省略**opt**参数，表示对所有符号进行检查。**

-incvar(name)　　**改变**INCLUDE**环境变量的名字为**name

-index(flags,ixtype,type(s))

此选项是对-strong**选项的补充，并与该选项联合使用。它指定**ixtype**为**type**类型数组或指针唯一合法的索引（下标）类型。**ixtype**和**type**都是用**typedef**定义的类型。**flags**取值为**c**或**d**。

c　　**表示除了**ixtype**还允许常量作为索引（下标）**

d　　**表示允许不用**ixtype**来指定数组的维数（大小）**

+libclass([all,angle,ansi,foreign]...)　　**设定头文件作为库头文件的判定条件**

angle　　**所有用“ <> ”括起来的头文件是库头文件**

foreign　　**所有在非当前目录的头文件是库头文件，注意：如果**#include**包含了一个完整的路径名，那么此头文件将不属于**foreign**类型。如果你想将此头文件当作库头文件处理，可以使用**<>**或者用**+libh

ansi　　**标准的**ANSI C/C++**库头文件，这些头文件作为库头文件，包括：**

assert.h	limits.h	stddef.h	ctype.h	locale.h
stdio.h	errno.h	math.h	stdlib.h	float.h
setjmp.h	string.h	fstream.h	signal.h	strsteam.h
iostream.h	stdarg.h	time.h		

all　　所有头文件都作为库头文件处理

缺省值为+libclass(angle , foreign)，注意这个选项不能积累，即只有最后一个才生效。

-/+libdir(directory[,...])　　减少或增加库路径，其中的所有头文件都不是或是库头文件。注意：即使+libdir(c:\compiler)，对于#include "C:\compiler\i.h"，仍然不当作库头文件处理，因为这里用了全路径名，没有进行搜索。此时只有对在Include目录中搜索出来的头文件才算是库头文件。

-/+libh(header[,...])　　减少或增加库头文件，此选项是可以积累的。

-library　　设置库源程序标志，例如：在开始部分用/*lint -library */说明的源程序（Module），将被当作库源程序（Library Module）

-limit(<n>)　　设置告警消息条数的上限(n<64000)，缺省没有限制

-/+lnt(ext)　　减少或增加以ext为扩展名的文件作为lnt文件对待处理

-lobbase(filename)　　建立Lob的基本文件用以在头文件很大时节省Lob文件的空间

例如：c1.cpp和c2.cpp都是比较小的，但是都包含了头文件gui.h，而gui.h文件很大，那么我们可以用如下命令：

lint -u c1.cpp -oo

lint -u -lobbase(c1.lob) c2.cpp -oo

这样，c2.lob文件中就只保存c1.lob中没有的东西，可以节省很多的硬盘空间。

-/+macros

增大宏的存储空间，一般情况下宏的最大存储空间为4096字节，每使用一次+macros将此上限乘以2，而使用一次-macros将使此上限除以2。这个选项必须在分析第一个模块文件时就进行设置，否则如果设置得太晚的话，将会导致设置无效。

-maxopen(<n>)　　设定文件可打开的最大次数

-mS -mD -mL -mP　　（内存）小模式，仅大数据，大数据和程序，仅大程序

-od[options](filename)　　输出声明（包括原型）到文件filename，+od表示append方式

options可以取值为：

f　　：　　仅仅输出函数

i　　：　　包括内部函数

s　　：　　包括结构定义

<width>　　：　　指定最大显示宽度

-oe(filename) 重定向stderr到文件filename，+oe表示append方式，可用于查看Help

-ol(filename) 输出库信息到文件filename

-oo[(filename)] 输出到LOB文件

-os(filename) 重定向stdout到文件filename，+os表示append方式，确保-os参数在要Lint的文件之前，否则会丢失Lint信息

-p[(n)] 仅进行预处理，可用于调试PCLint的预处理是否与预期一致，n表示最大输出宽度

-parent(Parent,Children) 增加一个父子关系到强类型层次树，其中父类和子类可以不是从同一个基本类型派生出来的。但是层次树中不允许出现循环。

-/+ppw(word[,...]) 减少或增加预处理的命令字，如+ppw(ident)将导致PCLint将#ident作为一个预处理命令字识别，并忽略该行而不报错（不认识）

+pragma(name,action) 联系 action 和 name，用法及含义不祥

-printf(#,id[,...]) 指定id等函数类似于printf函数，#为数值时表示该函数的格式在第#个参数中说明，第#个参数之后的参数被期望为在大小和类型上符合格式中的说明，同时，#中可以使用字符“ w ”，表示指针参数必须为far类型。如：

-printf(w2 , wsprintf)

-restore 恢复错误禁止状态

-/+rw(word[,...])

去掉或增加保留字，对于某些编译器特有的保留字，可以使用此选项让PCLint忽略该字。否则，PCLint会无法正确分析。（*ms）表示所有微软的关键字。例如：

unsigned char *restrict pch;其中restrict为某编译器特有的保留字。如果不做处理的话，PCLint将不能正确识别，告警pch没有定义。使用参数+rw(restrict)之后，PCLint能够正确忽略保留字并完成对变量pch的说明处理。

-save 保存当前的错误禁止设置

-scanf(#,id[,...]) 指定id等函数类似于scanf函数，其格式在第#个参数中说明，对这样的函数，第#个参数之后的参数要在类型和大小上与格式一致。

-sem(name,sem[,sem]...) 将一个语义集与函数联合起来

-size(flags,amount) 报告大集合体，flags为 a 表示auto变量，为 s 表示static 变量

-strong(Flags,Type(s))

每个Type类型作为一个带有Flags特性的强类型。注意，此选项必须在typedef之前生效。注意：强类型必须是用typedef定义的类型。Flags取值为：

A 在赋值给强类型时（赋值、返回值、参数传递和初始化）发出告警

i 忽略初始化

r 忽略Return语句

p 忽略参数传递

a 忽略赋值操作

c 忽略将常量赋值（包括整数常量、常量字符串等）给强类型的情况

z 忽略Zero赋值，Zero定义为任何非强制转换为强类型的0常量。例如：0L和(int)0都是Zero，但是(HANDLE)0当HANDLE是一个强类型的时候就不是Zero。(HANDLE *)0也不是。

J 当强类型与其它类型进行如下的二进制操作时进行检查

e 忽略==、!=和?:操作符

r 忽略>、>=、<和<=

o 忽略+、-、*、/、%、|、&和^

c 忽略该强类型与常量进行以上操作时的检查

z 忽略该强类型与Zero进行以上操作时的检查

X 当一个强类型的值被赋值给其它类型时发出告警

B 假定每一个Bolean类操作符都将返回一个与Type类型兼容的返回值，在所有需要判断Bolean值如if语句的地方都要检查结果是否符合这个强类型，否则告警。对于后半句主要是用于如下情况，if(a)...当a为int时，将产生告警，因为int与Bolean类不兼容，所以必须改为if(a != 0)...。

b 仅仅假定每一个Bolean类操作符都将返回一个与Type类型兼容的返回值

l 库标志，当强类型的值作为参数传递给库函数等情况下，不发告警

f 与B或b连用，表示抑止对1bit长度的位域是Boolean类型的假定，如果不选该项表示1bit长度的位域被缺省假定为Boolean类型。

如果Flags部分为空，表示所有Types都是强类型，但是除了对声明进行检查之外，不指定任何其它的检查。如果Types部分为空，表示除了用别的-strong选项说明的强类型之外的所有用typedef定义的类型都是强类型，且具有Flags属性。例如：

-strong(A) -strong(Ac,Count)或-strong(Ac,Count) -strong(A)都是表示对Count类型不做强类型检查，但是对于其它用typedef定义的类型都是强类型并进行赋值检查。

-t# 设置Tab键的大小为#，缺省值为8

-u 单元检查，抑止许多模块间问题告警，如526、552等

-unreachable

表明程序中的一个点是不可达的，用于抑止某些告警，如：

```
int f(n)
{
  if (n) return n;
  exit(1);
  //lint -unreachable
}
```

此处用以防止PCLint认为exit语句后执行了一个隐含的return，而隐含的return一般是不返回值的，这就会出问题，因为声明要返回值int。所以使用此选项抑止此告警。

-u<name>　　取消对name的定义，对后续的模块文件生效

--u<name>　　忽略以前及以后对name的定义

-w<level>　　设置告警级别，取值范围(0,1,2,3,4)

-w0　无任何消息（致命错误Fatal Errors除外）

-w1　仅错误消息（Errors），无告警（Warnings）和提示（Informationals）

-w2　仅错误消息和告警消息

-w3　错误、告警和提示消息（这是缺省值）

-w4　所有消息

-wlib(<level>)　　设置对库文件的告警级别，取值范围及含义同上，缺省级别为4

-wprintf(#,id[,...])　-printf选项的宽字符（wide char）版本

-wscanf(#,id[,...])　-scanf选项的宽字符（wide char）版本

-zero[(#)]　　对于所有错误号大于#的，都设置退出码为0，这在用make文件时很有用

-$　　允许标志符中使用$作为标志符的一部分

7 编译器相关选项

-a#<predicate>(tokens)　　使 #predicate (tokens)为True，用于Unix

-d<name>()[=<value>]　　用于定义类似于函数的宏

-#d<name>[=<value>]　　定义标号，仅用于 #include

举例如下：

-#dtime=Filename　不会影响time在非Include以外的地方作为标志符使用

#include　time　某些VAX-11 C可以这样使用Include

-overload(X)　　X为16进制常数，用于设置标志位，缺省值为7

bit 1　内存模式满足优先于ANSI标准满足。例如：int n; f(&n);将选择void f(int const *)而不选择void f(int far *)，因为内存模式far不满足。

bit 2　内存模式及ANSI标准同时满足优先于单个满足。

bit 4　内存模式对于引用参数（&）有意义。

设置时，如果要设置哪几个位，X就等于位数和。如：7 = 1 + 2 + 4，表示同时置为bit 1、2和4。其余依此类推。此选项影响函数重载（Overload）的选择。

-plus(Char)　　将字符Char作为“+”对待，用于解决某些操作系统上不方便使用+的替代

-template(X)　　X为16进制常数，用于设置对模板处理的标志位

bit 1　当前只有这一个标志位，缺省是关的。表示对于模板基类采用积极的处理方式。一般来说，模板基类直到实例化的时候才进行处理。但是对于某些库，尤其是STL库，基类必须采用积极的处理方式。因为它们提供了模板处理时需要的名字。

_bit　　1个bit宽的数据类型，使用+rw(_bit)激活

_gobble　　用于忽略它本身和其后的下一个单词，使用+rw(_gobble)激活

_to_brackets	用于忽略它本身和其后用各种括号（()、[]和{}）括起来的部分，使用+rw(_to_brackets)激活
_to_semi	用于忽略它本身及其后的所有东西直到遇到分号（;）为止（包括分号），使用+rw(_to_semi)激活

以上的三个选项，其用法相似，举例如下：

```
-dinterrupt=_to_brackets    //令interrupt等同于_to_brackets
+rw(_to_brackets)           //将_to_brackets作为关键字激活，而interrupt是等同于该关键字的，将导致interrupt及其后用括号括起来的部分Lint时被忽略
```

结果如下，将导致下面的语句被忽略：

```
interrupt(3)
interrupt[5,5]
interrupt{x,x}
```

8 各种使用说明

8.1 库模块文件的使用（Library Modules）

库模块文件用来描述函数（非原型）的参数列表，对于库模块文件中声明的任何对象（不是指C++的对象，包括结构等）都不需要在其它地方定义和使用，这一点和库头文件一样。它的目的是为了使Lint能够处理源程序所包含的非原型的库函数。即使对于ANSI标准的编译器，有时候我们也会使用到非原型的库，这时就需要使用库模块文件了。举例如下：

假设你被提供了一个图形库的目标文件g.lib和头文件g.h。如果g.h包含原型，你不需要使用库模块文件。如果g.h不包括原型，而你又有该库的源程序g1.c－g25.c的话，你可以用如下命令生成原型描述。

```
lint -u g*.c -od(gproto.h)
```

这个命令将生成所有函数和数据对象的原型到文件gproto.h中，其中不包括结构定义，如果你的g.h文件中没有结构定义的原型，你还需要使用-ods命令将结构定义也生成原型。然后你可以定义你的库模块文件如下：

glib.c：

```
/*lint -library */
#include "g.h"
#include"gproto.h"
```

接下来你就可以用它来Lint包含该库的源程序program了，命令如下：

```
lint co glib program        //co为编译器的配置文件（lnt）
```

为了减少处理时间，你还可以先将glib.c生成为LOB文件以加快处理。如下：

```
lint -u co glib -oo(glib.lob)
lint co glib.lob program
```

8.2 汇编（非C、C++）文件的处理

如果你的工程中包含汇编代码或其它非C、C++代码，你必须如下处理以使得Lint不会因为这些缺失的代码产生误告警。最常用的方法是创建一个头文件来描述汇编部分的代码，这个头文件必须被当作库头文件处理，以免PClint对其中的声明产生告警（因为无法得到声明的定义和使用部分的代码）。因此我们必须这样使用：+libh(asm.h)。

另外如果一个变量alpha仅在汇编代码部分被使用，Lint时会告警552，我们可以使用如下命令抑止该告警：-esym(552,alpha)。需要说明的是，这个选项不能放在asm.h文件中，因为库头文件中的选项只有在第一次该头文件被包含时才会生效，那么这个选项对于前面处理过的源文件就是无效的。所以该命令应该包含到配置文件中。

8.3 强类型

8.3.1 应用举例：

例一：

```
//lint -strong( AJXb , Bool )          //由于没有选择f，因此1bit字位缺省为Boolean类型
//lint -strong( AJX , BitField )
typedef     int             Bool;
typedef     unsigned        BitField;
struct foo
{
    unsigned a:1 , b:2 ;             //成员a和c都被缺省假定为Bool类型，b不是强类型
    BitField c:1 , d:2 , e:3;        //成员d和e都是BitField类型
}x;                                  //这个例子也说明了一个变量只能是一种强类型，如c。
void f()
{
    x.a     =       (Bool) 1;       //OK，同一类型赋值
    x.b     =       (Bool) 0;       //NO，违反X规则
    x.a     =       0;              //NO，违反A规则
    x.b     =       2;              //OK，b不是强类型
    x.c     =       x.a;            //OK，同一类型赋值
    x.e     =       1;              //NO，违反A规则
    x.e     =       x.d;            //OK，同一类型赋值
}
```

例二：

```
//lint -index( d , Count , Temperature )
typedef     float     Temperature;
```

```
typedef     int     Count;
Temperature         t[100];         //OK，因为d允许使用非Count类型说明数组大小
Temperature         *pt = t ;
Count       i;

t[0]        =       t[1];           //NO，因为没有c，所以不允许使用常数作为下标
for( i=0 ; i<100 ; i++)
    t[i] = 0.0;                     //OK，i是Count类型
pt[1] = 2.0;                        //NO，因为没有c，所以不允许使用常数作为下标
i = pt - t ;                        //OK，pt - t 的结果是Count类型
```

说明：如果要对多维数组进行强索引类型检查，应该使用递归式定义数组方式，以二位数组 Screen[25][80]**的定义方式举例如下：**

```
/*lint      -index( d , Row_Ix , Row )
            -index( d , Col_Ix , Att_Char)     */

typedef     unsigned short  Att_Char;
typedef     Att_Char        Row[80];//Row是一个有80个元素的数组，每个元素类型为Att_Char
typedef     Row             Screen[25];
typedef     int             Row_Ix;
typedef     int             Col_Ix;
```

8.3.2 类型层次

由于一些函数需要处理一类数据类型，因此为了方便处理需要提出类型层次的概念。在类型层次中，父类型和子类型可以相互赋值。这样，我们处理一类相似的子类型时可以用父类型进行声明，这是很方便的。编程人员也是这么做的。请看windows.h**中的**WORD、HANDLE**等声明。类型层次可以有任意多层。请看下面的例子：**

```
//lint  -strong(AJX)
typedef     unsigned        Flags;  //父类
typedef     Flags           Flags1; //子类
typedef     Flags           Flags2;
#define     FZERO           (Flags) 0
#define     F_ONE           (Flags) 1
void m()
{
    Flags1 f1 = FZERO;              //OK，父类可以赋值给子类
    Flags2 fs;
    f2 = f1;                        //NO，两个子类之间不能互相赋值，AX
```

```
    if( f1 & f2 )                    //NO，两个子类之间不能进行逻辑操作，J
        f2 = f2 | F_ONE;             //OK，父类和子类可以进行逻辑操作
    f2 = F_ONE | f2;                 //OK，且父类和子类逻辑操作的结果为子类类型
    f2 = F_ONE | f1;                 //NO，见上一个说明
}
```

注意：强类型检查并不会抑止其它的检查，如缺失精度等。一旦一个类型被说明为强类型，就无法再改变它的强类型属性，也不能变成非强类型。

8.4 PCLint的预处理符

_lint　　Lint过程中，_lint为真。可以用其使某段代码不被Lint。可能是因为你不想Lint该段代码，也可能该段代码是用汇编等其它语言编写的，无法Lint。使用方式如下：

```
#ifndef    _lint
.....
#endif
```

8.5 选项的处理顺序

命令行中的选项是从左到右顺序处理的，如：

lint alpha　beta　-idirectory

那么在Lint文件alpha和beta时并没有包含directory目录，因为-i命令在最后才会处理。

8.6 使告警最大化

如果我们定期的对新程序进行Lint，并且对于每次Lint的结果修改错误，以使告警不再出现。那么，我们就可以使用以下措施，使告警尽可能的多，以保证代码质量尽可能的好。

+fsc　+fxa　+fxc　+fxf　+fxs　+fpn　-strong(AJX)　-w4

9 附录：PCLint在Source Insight中的使用

如果你在Source Insight打开上次保存的Lint结果文件，那么如何使其与源文件链接起来以方便使用呢？可以使用“Search->Parse Source Links”建立Lint结果文件与源程序的链接。当然你应该首先打开相应的工程。一般在Pattern编辑框中输入：^\(.*\.[a-zA-Z]\)\w\([0-9]+\).*，然后选择“File,then Line”即可完成源文件链接功能。对于无法链接的情况，可以根据你的Lint结果文件的格式，按照下面的规则修改Pattern即可。

9.1 Source Insigh的正规表达式

字符	匹配
^ (仅用于开头)	一行的开始
.	任何单个字符
[abc]	集合abc中的任何单个字符
[^abc]	任何不属于集合abc的单个字符，^的作用域为[]之间所有

*	前面字符的0次或多次重复
+	前面字符的1次或多次重复
\t	Tab字符
\s	空格字符（Space字符）
\w	空白字符（Tab或Space字符）
$	一行的结束
\	恢复此表中特殊字符的原意，如：a*b，表示匹配字符串a*b，而不是匹配0或多个a后接一个b
\(和 \)	它们包含的部分将作为一个组，一个正规表达式中的每个组将有一个编号，从1开始。主要用于替换操作

以下是集合的使用及含义：

集合类型	例子	含义
[<character list>]	eg. [abcde]	匹配集合内的任一字符，集合长度不限
[x-y] x<y	eg. [a-z]	匹配x到y之间（包含x和y）的任一字符，
组合使用方式	eg. [WXYa-z0-9]	