笔记以GB/T 15272-94为基础，该标准基本是C89的中文翻译。

# 笔记

Definition 定义 declaration

Implementation 实现

Convention 约定

Syntax 语法

Constraints 约束

Parameter 形参 函数定义的

Argument 实参 调用输入的

Compliance 一致性

int main(int argc, char \*argv[])

argv[0] 表示程序名

argv[1] – argv[argc-1] 程序参数

易变型 volatile

Space 空格

Control character 控制字符

Horizontal tab 横向制表

Vertical tab 纵向制表

Form feed 换页

显示

\a 告警

\b 退格

\f 换页

\n 新行

\r 回车

\t 横向制表

\v 纵向制表

## 0-4 总共才3页

## 5 模型

这部分总共9页。

翻译限定值 5.2.4.1

---- 这里定义了嵌套层数、标识符个数限制等，函数参数个数（31个）

比如：

单个结构体运行有127个成员

C89规定函数参数个数限制为32个，而c99规定函数参数个数限制为127个

|  |  |  |  |
| --- | --- | --- | --- |
|  | C89 | C99 | C11 |
| 函数参数个数 | 32 | 127 | 127 |
| 一行字符数 | 509 | 4095 | 4095 |
| 结构体成员数 | 127 | 1023 | 1023 |
| 内部标识符有效字符数 | 31 | 63 | 63 |
| 外部标识符有效字符数 | 6 | 31 | 31 |
| C语言指针级数 | 12 | 12 | 12 |

## 6 语言

总共60页。

### 6.1 词法元素

6.1.1 关键字

|  |  |  |
| --- | --- | --- |
| C89 | C99 | C11 |
| 6.1.1 32个  auto double int struct  break else long switch  case enum register typedef  char extern return union  const float short unsigned  continue for signed void  default goto sizeof volatile  do if static while | 6.4.1 37个（新增5个）  restrict \_Bool \_Complex  \_Imaginary inline | 6.4.1 44个（新增7个）  \_Alignas \_Alignof \_Atomic  \_Generic \_Noreturn  \_Static\_assert \_Thread\_local |
|  |  |  |

标识符构成规范：非数字字符（下划线、大小字母）和数字构成，且开头为非数字。

标识符作用域

类型：基本类型，派生类型：数组、结构、联合、函数、指针

算术符号 operators : [ ] ( ) . -> ++等

标点符号 punctuators: [ ] ( ) \* , : #

头文件 #include

6.1.8 preprocessing 预处理

6.1.9 注释 comments

/\*\*/

C99引入 // 6.4.9

### 6.2 conversions 转换

将操作数的值从一种类型转换为另一种类型

### 6.3 表达式 expressions

### 6.4 常量表达式 constant expressions

### 6.5 声明 declarations

存储类区分符：

typedef extern static auto register

类型限定词：const volatile

### 6.6 语句 statements

带标号语句

复合语句

表达式语句

选择语句

循环语句

跳转语句

### 6.7 外部定义

### 6.8 预处理指示

#if

#ifdef

#ifndef

#elif

#else

#endif

#include

#define

#undef

#line

#error

#pragma

1、#Pragma message参数能够在编译信息输出窗口中输出相应的信息，比如：

#ifdef \_X86

#pragma message(“\_X86 macro activated!”)

#endif

2、#pragma code\_seg能够设置程序中函数代码存放的代码段

3、#pragma once (比较常用)若用在头文件的最开始处就能够保证头文件被编译一次

还有好多

6.8.3.3 ##

连接字符串的作用：

#define def\_u32\_array(\_\_name, \_\_size) uint32\_t array\_##\_\_name[\_\_size];

def\_u32\_array(sample\_buffer, 64) 展开后实际是：

uint32\_t array\_sample\_buffer[64];

可变参数宏？

在ANSI-C99标准引入可变参数宏的时候，又贴心了加了一个不那么起眼的语法：当下面的组合出现时 ",##\_\_VA\_ARGS\_\_"，如果\_\_VA\_ARGS\_\_是一个空字符串，则前面的","会一并被删除掉。

6.8.8 预定义的宏名

\_\_LINE\_\_

\_\_FILE\_\_

\_\_DATE\_\_

\_\_TIME\_\_

\_\_STDC\_\_

标准文档中没有找到如下宏定义？

[\_\_FUNCTION\_\_](https://link.zhihu.com/?target=https%3A//www.codersrc.com/archives/9341.html) 程序预编译时预编译器将用所在的函数名，返回值是字符串;

## 7 库 library

/usr/lib/libc.a 和/或 /usr/lib/libc.so

使用尖括号表示在包含文件目录中去查找(包含目录是由用户在设置环境时设置的)

使用双引号则表示首先在当前的源文件目录中查找，若未找到才到包含目录中去查找。

linux下，上述标准路径有：/usr/include，/usr/local/include

编译：

通过参数-I指定头文件的搜索路径

通过环境变量C\_INCLUDE\_PATH/CPLUS\_INCLUDE\_PATH/OBJC\_INCLUDE\_PATH

再找内定目录

/usr/include

/usr/local/include

/usr/lib/gcc-lib/i386-linux/2.95.2/include

链接：

LIBRARY\_PATH

运行：

LD\_LIBRARY\_PATH

libc是Linux下原来的标准C库，也就是当初写hello world时包含的头文件#include < stdio.h> 定义的地方，后来逐渐被glibc取代。glibc最主要的功能就是对系统调用的封装，glibc自身也提供了一些上层应用函数必要的功能,如string,malloc,stdlib,linuxthreads,locale,signal等等。

总共约60页。

|  |  |  |
| --- | --- | --- |
| C89 | C99 | C11 |
| 共15个  Assert.h  Ctype.h  Errno.h  Float.h  Limits.h  Locale.h  Math.h  Setjmp.h  Signal.h  Stdarg.h  Stddef.h  Stdio.h  String.h  Time.h | 共24个，新增9个：  Inttypes.h  Complex.h  Iso646.h  Stdbool.h  Tgmath.h  Fenv.h  Stdint.h  Wchar.h  Wctype.h | 共29个，新增5个：  Sdtnoreturn.h  Stdalign.h  Threads.h  Stdatomic.h  Uchar.h |

|  |  |
| --- | --- |
|  |  |
| Errno.h | 感觉没什么用  EDOM  ERANGE |
| Assert.h | assert |
| Ctype.h | Isdigit  Islower  Isspace  Issupper  Tolower  toupper |
| Setjump.h | 感觉没怎么用？  Setjmp  longjmp |
| Signal.h | Signal  raise |
| Stdarg.h | 变长实参  va\_list  va\_start  va\_arg  va\_end |
| stdio.h | 文件操作 |
| stdlib.h | malloc free system |
| Sting.h | Memcpy  Strcpy  Strcat  Memcmp  Strcmp  Strchr  Strtok  Memset  strlen |
| Time.h | Mktime  time |

## 8 附录

总共约57页。

# 其他

## static

## const

1. 修饰变量：被它修饰的变量值不能再改变, const在修饰变量的时候一定要初始化，否则之后就不能再进行赋值了。

const int a;

int const a;

2、修饰指针

（1）常量指针

const int \*p; 不能通过这个指针改变变量的值，常量指针可以指向其他的地址

也可以写为: int const \*p;

（2）指针常量

int x = 8;

int\* const p = &x; 指针本身是一个常量不可以再指向其他地址

（3）指向常量的常量指针

int x = 8;

const int\* const p = &x; 指针指向的位置不能改变，指向的值也不能改变

3、修饰函数参数

函数参数也可以用const来修饰，防止函数修改参数中的内容

函数的返回值也可以用const来修饰，不过意义不是很大

const int\* fun1(const int\* param)

4、修饰全局变量

防止不必要的修改

## volatile

关键字提醒编译器它所定义的变量随时都可能改变，因此编译后的程序每次需要存储和读取这个变量的时候，都对直接从这个地址中读取数据。

如果没有volatile关键字，则编译器可能优化存储和读取（比如将内存变量缓存到寄存器），如果这个变量由别的程序更新的话，将出现不一致的现象。如果用这个关键字声明变量，编译器对访问该变量的代码就不再进行优化，从而可以稳定地访问特殊的地址。

## 指针

指针这个词，它就是地址变量，里面装着另一个变量的地址

指针和数组

指针作为数组来使用

数组当成指针

指针和[const](https://www.zhihu.com/search?q=const&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22answer%22%2C%22sourceId%22%3A2572929194%7D)

指向数组的指针

指针数组

数组指针

## typedef

## #pragma once

为了避免同一个头文件被包含（include）多次，C/C++ 中有两种宏实现方式：一种是 #ifndef 方式；另一种是 #pragma once 方式。

#pragma once 不支持跨平台！#ifndef 支持跨平台！