C语言深度剖析 石虎

第一章 关键字

绪

定义和声明最重要的区别是；定义创建了对象并为这个对象分配了内存

声明没有分配内存

1.1 auto

1.2 register

1.3 static

作用一： 修饰变量。

1.静态全局变量

作用域仅限于变量被定义的文件中。其他文件即使用extern声明也没法使用他。在定义前面的代码行也不能使用它，想要使用就得在前面加extern。介意直接在文件顶端定义静态全局变量。

2.静态局部变量

在函数体内部定义的变量，只能在该函数里使用。和静态全局变量一样，总是存在内存的静态区。变量的值在函数运行结束也不会被销毁。函数下次使用仍然是这个值。



作用二： 修饰函数。

函数前加static使得函数成为静态函数。此处的static不是指存储方式，而是指函数的作用域仅限于本文件（又称为内部函数）。不用担心自己定义的函数是否会与其他文件中的函数同名。

1.4.1 基本数据类型



1.4.2 数据命名规则

标识符的一般规则：

1.命名只管可拼读，望文生义，便于记忆。

2.命名长度应符合最小长度和最大信息的原则。英文尽量不缩写。

3.标识符由多个词组成时，每个词的第一个字母大写，其余全部小写。

4.除了驱动开发的管脚命名等，尽量避免出现数字编号。

5.多文件时全局变量或函数要加范围限定符。

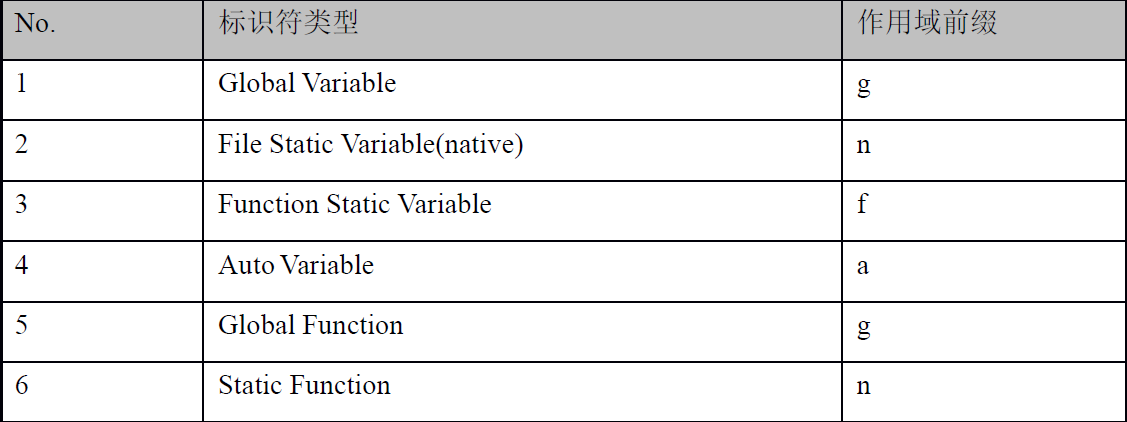
标识符的命名规则：

6.标识符分为两部分：规范标识符前缀+含义标识。非全局变量可以不使用范围限定符前缀。

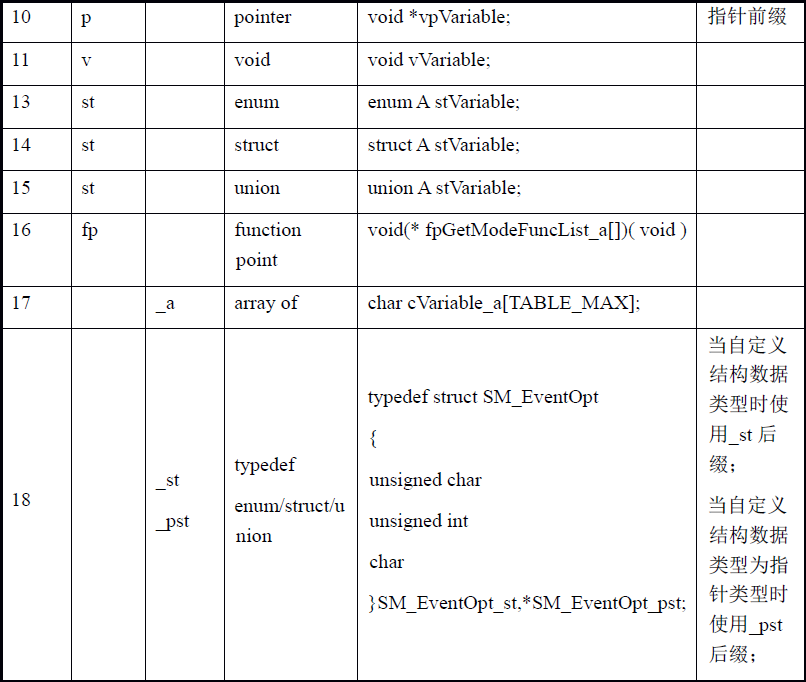
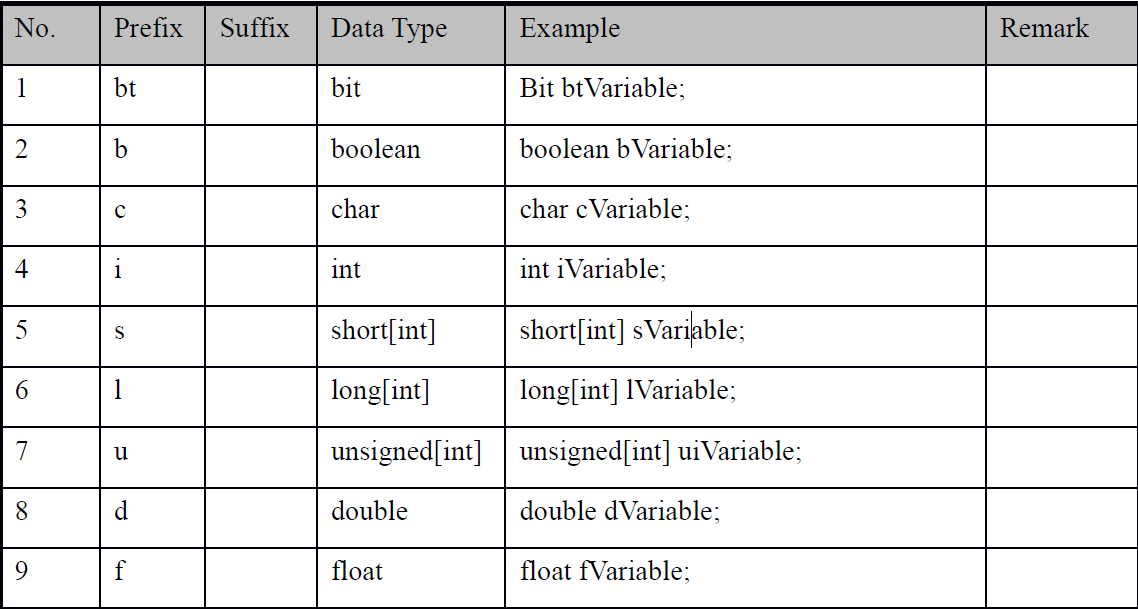
|-标识符前缀-|

模块名缩写 \_ 作用域前缀 数据类型前缀 [指针前缀] 含义标识 数组/结构后缀

作用域前缀



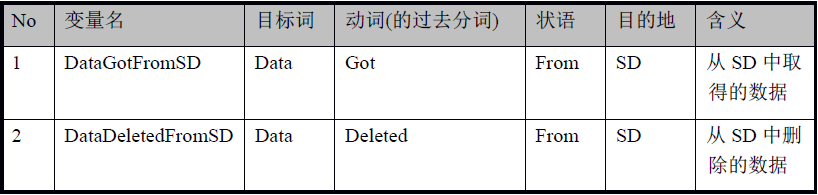
数据类型前缀



含义标识

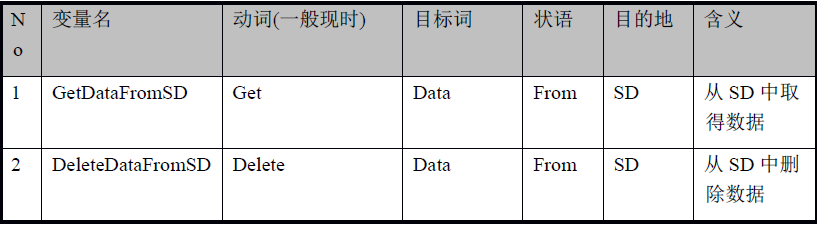
1）变量命名使用名词性词组

变量含义标识符构成：目标词 + 动词（的过去分词）+ [状语] + [目的地]



2）函数命名使用动词性词组

函数含义标识符构成： 动词（一般现在时）+ 目标词 + [状语] + [目的地]



7.程序中不得出现仅靠大小写区分的相似标识符。

8.一个函数名禁止被用于定义其他变量名等。

9.所有宏定义、枚举常数、只读变量全用大写字母命名，用下划线分割单词。

10.定义变量的同时千万千万别忘了初始化。定义变量时编译器并不一定清空了这块内存，它的值可能是无效数据。

11.不同类型的数据之间的运算要注意精度扩展问题。

1.5 sizeof关键字



1.6 signed unsigned

1. 一个32位的signed int类型整数其值表示法范围为：- 231～231 -1；

2. 8位的char类型数其值表示的范围为- 27～27 -1;

3 一个 32位的 unsigned int类型整数其值表示法范围为：0～ 232 -1；

4. 8位的 char类型数其值表示的范围为 0～28 -1。

5. 同样我们的 signed 关键字也很宽恒大量，你也可以完全当它不存在，编译器缺省默认情况下数据为signed 类型的。

数值一律用补码来表示（存储）。主要原因是使用补码，可以将符号位和其它位统一处理；同时，减法也可按加法来处理。正数的补码与原码相同，求负整数的补码，将其对应正数二进制表示所有位取反（0变1，1变0）后加1 。这种方式最后需要加上最前面的符号位。

如果是char型数据

负数 正数二进制 取反 +1 补码（2的8次-负数的绝对值）

-1 00000001 11111110 11111111 11111111(0xff)

-2 00000010 11111101 11111110 11111110(0xfe)

.

.

-127 01111111 10000000 10000001 10000001(0x81)

-128 10000000 01111111 10000000 10000000(0x80)

-129 10000001 01111110 101111111 01111111(0x7f)

-256 100000000 011111111 1100000000 00000000(0x00)

注；红色的是符号位

-129因为正数取反+1 后为01111111，八位有符号数不能是-129，发生了溢出，最高位符号位1丢失，剩下的8位为0x7f。

（-1）+（-127）

11111111 （-1）

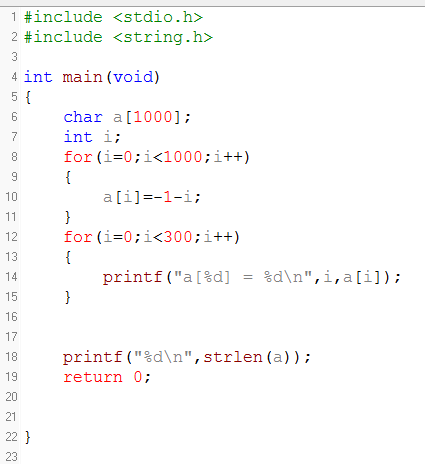
10000001 （-127）

------------------

1(舍弃)10000000 （-128） 存储仍然是补码形式

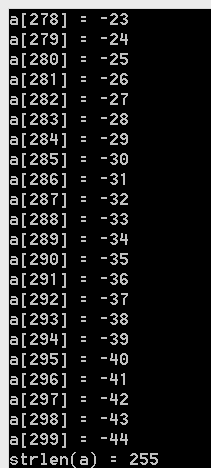
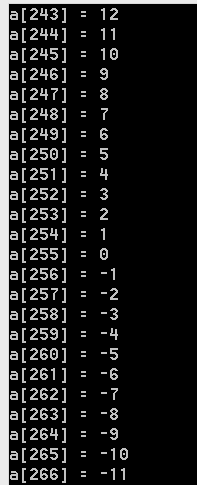
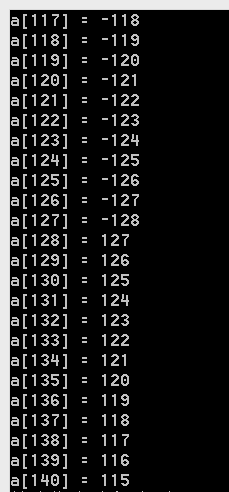
0：0000 0000  
-1：1111 1111（总之-1加1溢出后是0）  
-2：1111 1110（总之-2加1是-1）

-127：1000 0001(总之-127+127溢出后是0)  
-128：1000 0000（总之-128+1是-127）  
即-128加上127再加1溢出后是0——C/C++里存放整数的统一性。



分析：a[i] 从 -1 --------> -128

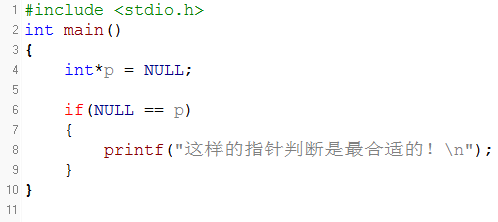
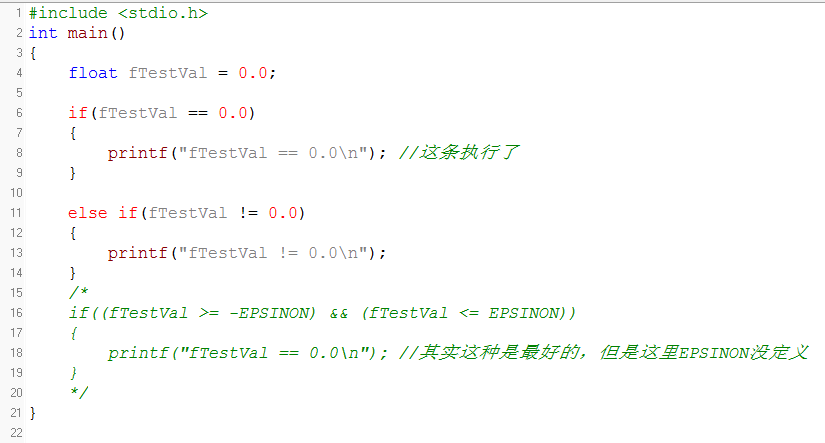
-129溢出，溢出以后的值是127，所以又从 127 -------> 0



strlen 函数是计算字符串长度的，并不包含字符串最后的‘\0’。而判断一个字符串是否结束的标志就是看是否遇到‘\0’。如果遇到‘\0’，则认为本字符串结束。



1.7 if else



1.7 switch case

1.if、else 一般表示两个分支或是嵌套表示少量的分支，但如果分支很多的话……还是用switch、case 组合。

2.最后必须使用default 分支。即使程序真的不需要default 处理，也应该保留语句。

3. case 后面只能是整型或字符型的常量或常量表达式（想想字符型数据在内存里是怎么存的）。

4.按字母或数字顺序排列各条case 语句。

5.把正常情况放在前面，而把异常情况放在后面。

6.按执行频率排列case 语句。

7.简化每种情况对应的操作。。一般来说case语句后面的代码尽量不要超过20行。

8.不要为了使用case 语句而刻意制造一个变量。

9.把default 子句只用于检查真正的默认情况。

1.8 do while for

1.在多重循环中，如果有可能，应当将最长的循环放在最内层，最短的循环放在最外层，以减少CPU跨切循环层的次数。

2.半开半闭区间写法和闭区间写法虽然功能是相同，但相比之下，半开半闭区间写法写法更加直观。

3.不能在for 循环体内修改循环变量，防止循环失控。

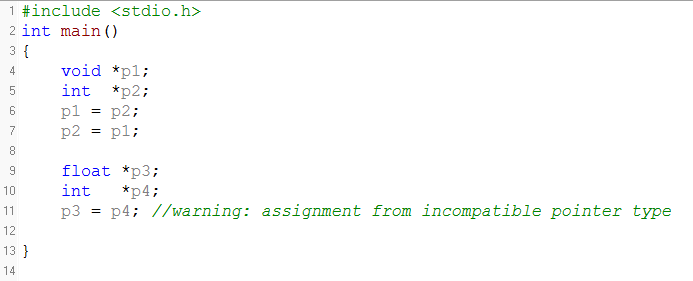
4.循环要尽可能的短，要使代码清晰，一目了然。

5.把循环嵌套控制在3 层以内。

1.9 void 和 void\*

1. void 的字面意思是“空类型”，void \*则为“空类型指针”，void \*可以指向任何类型的数据。

2.任何类型的指针都可以直接赋值给void\*，无需进行强制类型转换。



3.如果函数没有返回值，那么应声明为void 类型。

4.千万小心又小心使用void 指针类型。

5.如果函数的参数可以是任意类型指针，那么应声明其参数为void \*。

例如：void \* memcpy(void \*dest, const void \*src, size\_t len);

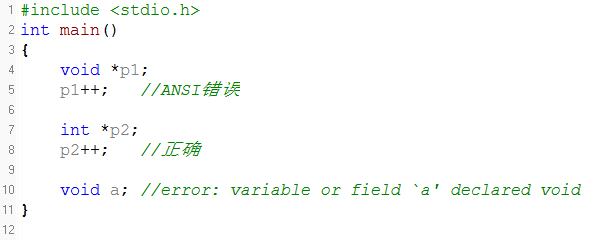
void \* memset ( void \* buffer, int c, size\_t num );

这样，任何类型的指针都可以传入memcpy 和memset 中，这也真实地体现了内存操作

函数的意义，因为它操作的对象仅仅是一片内存，而不论这片内存是什么类型。如果memcpy和memset 的参数类型不是void \*，而是char \*，那才叫真的奇怪了！这样的memcpy 和memset明显不是一个“纯粹的，脱离低级趣味的”函数！

memcpy 和memset 函数返回的也是void \*类型，标准库函数的编写者都不是一般人。

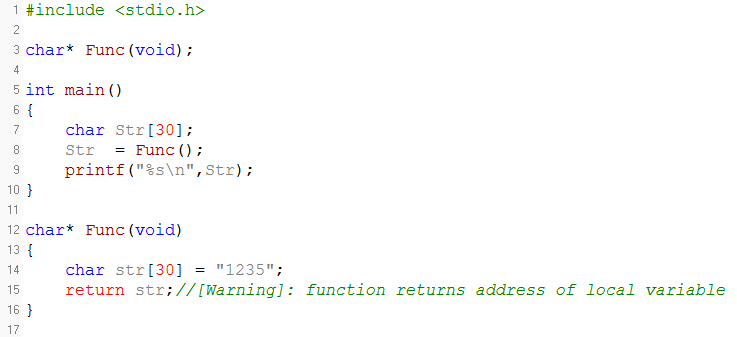
6. void 不能代表一个真实的变量。



1.10 return

1. return 语句不可返回指向“栈内存”的“指针”，因为该内存在函数体结束时

被自动销毁。



1.11 const

1.const修饰的只读变量

//精确的说应该是只读的变量，其值在编译时不能被使用

//因为编译器在编译时不知道其存储的内容。

//const推出的初始目的正是为了取代预编译指令，消除它的缺点，同时继承它的优点。

//编译器通常不为普通const 只读变量分配存储空间，而是将它们保存在符号表中

//这使得它成为一个编译期间的值，没有了存储与读内存的操作，使得它的效率也很高。

2.节省空间，避免不必要的内存分配，同时提高效率

#define M 3 //宏常量

const int N=5; //此时并未将N放入内存中

int i=N; //此时为N分配内存，以后不再分配！

int I=M; //预编译期间进行宏替换，分配内存

int j=N; //没有内存分配

int J=M; //再进行宏替换，又一次分配内存！

const定义的只读变量从汇编的角度来看，只是给出了对应的内存地址，而不是象#define

一样给出的是立即数，所以，const 定义的只读变量在程序运行过程中只有一份拷贝（因为它是全局的只读变量，存放在静态区），而#define 定义的宏常量在内存中有若干个拷贝。#define 宏是在预编译阶段进行替换，而const 修饰的只读变量是在编译的时候确定其值。#define 宏没有类型，而const 修饰的只读变量具有特定的类型。

3.修饰变量

int const k = 2;

const int m = 3; //两种写法都可以

int const a[5]={1, 2, 3, 4, 5};

const int b[5]={1, 2, 3, 4, 5};

const int \*p; // p可变，p指向的对象不可变

int const \*p1; // p可变，p指向的对象不可变

int \*const p2; // p不可变，p指向的对象可变

const int \*const p3; //指针p和p指向的对象都不可变

先忽略类型名（编译器解析的时候也是忽略类型名），我们看const 离哪个近。“近水楼

台先得月”，离谁近就修饰谁。

const \*p; //const 修饰\*p,p 是指针，\*p 是指针指向的对象，不可变

const \*p; //const修饰\*p,p 是指针，\*p 是指针指向的对象，不可变

\*const p; //const修饰p，p 不可变，p 指向的对象可变

const \*const p; //前一个const 修饰\*p,后一个const 修饰p，指针p 和p 指向的对象

都不可变

4.修饰函数

const 修饰符也可以修饰函数的参数，当不希望这个参数值被函数体内意外改变时使

用。例如：void Fun(const int i);

告诉编译器i 在函数体中的不能改变，从而防止了使用者的一些无意的或错误的修改。

5.修饰函数的返回值

const 修饰符也可以修饰函数的返回值，返回值不可被改变。例如：

const int Fun (void);

在另一连接文件中引用const 只读变量：

extern const int i; //正确的声明

extern const int j=10; //错误！只读变量的值不能改变。

注意这里是声明不是定义，关于声明和定义的区别，请看本章开始处。