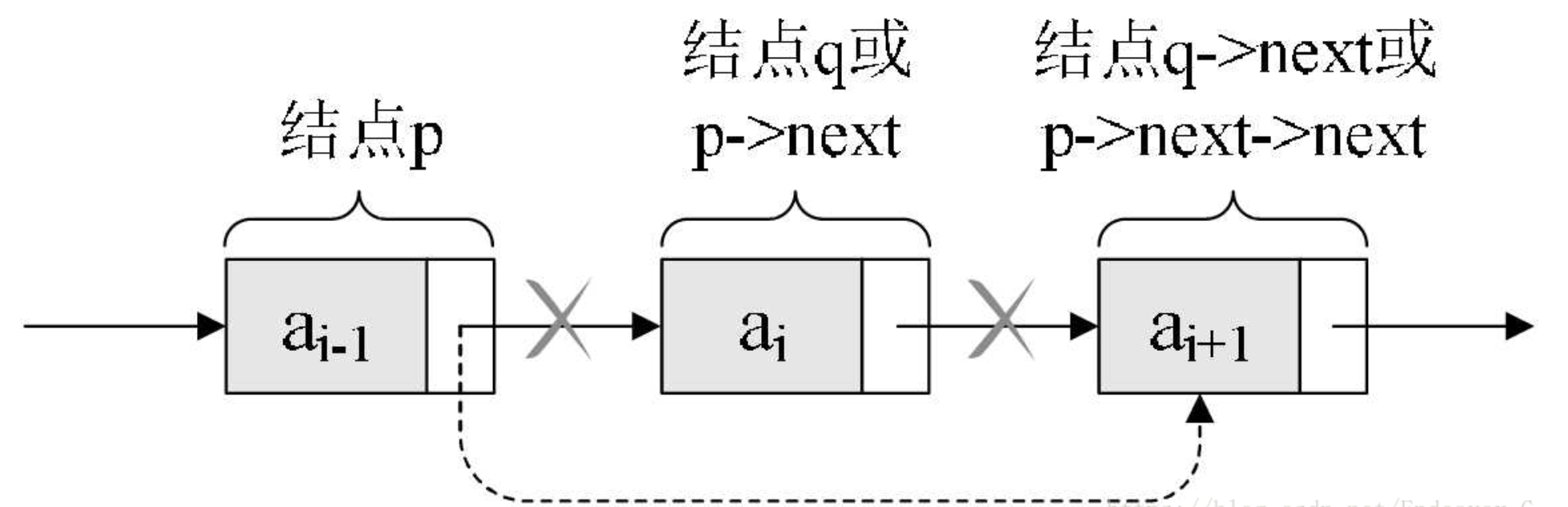
# 关于C 语言指针的用法

我在批改同学的实验报告时，发现一个普遍的问题。同学们程序不太规范，指针和数组使用。设计数据结构等问题，下面简单举例说明一下。

1. 函数的规范写法

以删除链表一个节点为例：



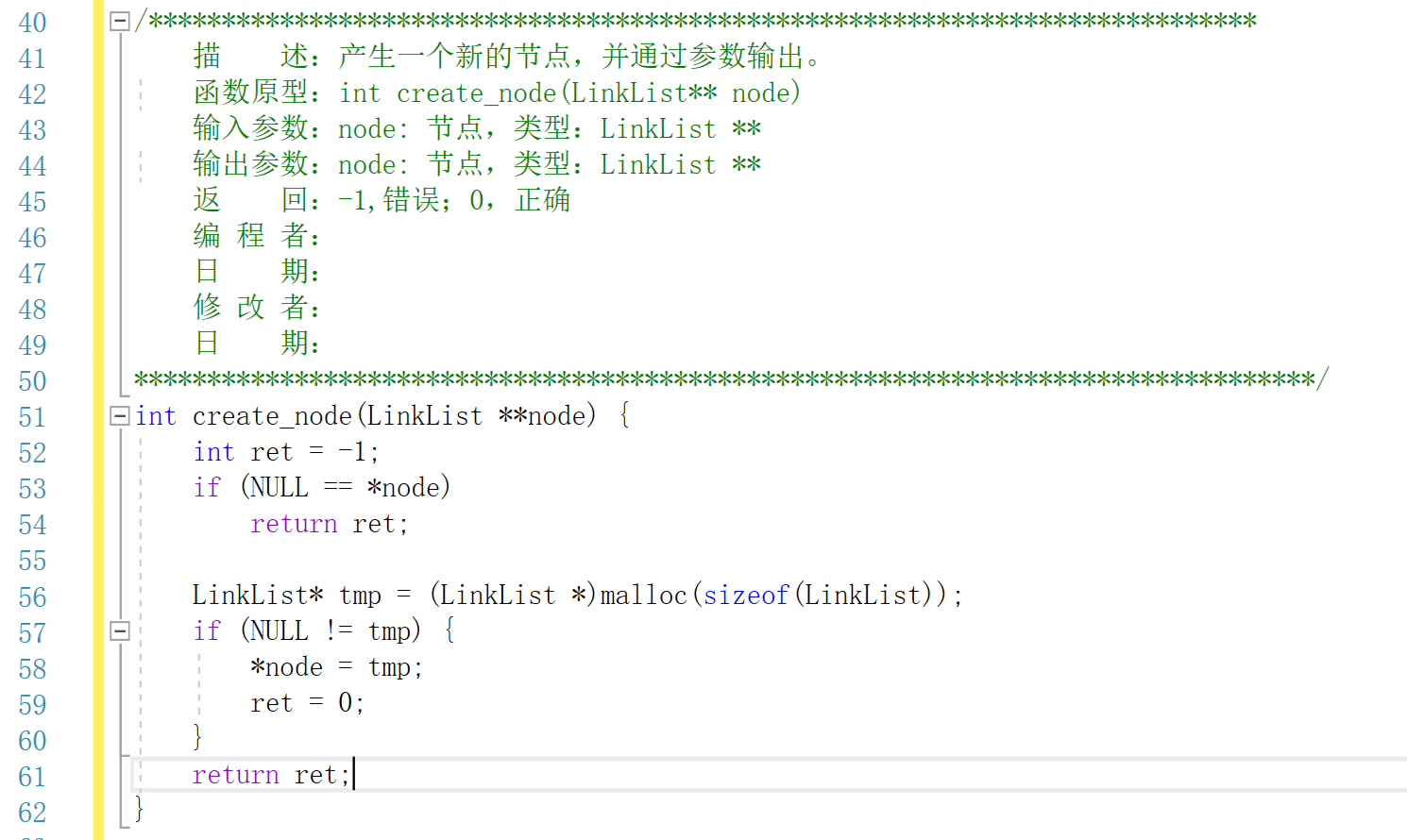


**说明：**

1. **行 1~12 ： 函数注释。**
2. **行 15~16： 输入参数检查，每一函数必须对入口参数检查**
3. **行17： 断言，必须保证括号中的表达式为 true, 在release版本中这句不起作用**
4. **行18，28： 空一行使程序阅读清晰，同时是一个小功能分块**
5. **行15：防止少写一个‘=’，C 语言在 if( list = NULL) 不会出错，但程序就出问题了。**
6. **行31~32： 这是习惯写法，free(p); p=NULL;非常必要，下面还会说明。**
7. 多级指针的使用,

不要无意义地使用多级指针。比喻 void delete(LinkList \*\*list,int n)这样设计函数毫无意义，反而造成不必要的复杂和麻烦。在链表、队列、栈、树和图数据结构中，操作的函数很少用到二级指针。而我们的作业大多数都用到了二级指针？？？

1. 函数内申请一个地址或改变指针的指向，并通过参数输出时，必须用双指针；



用法：

……

**LinkList \*node = NULL;**

**int ret = create\_node(&node);**

**/\* 这时，node 不为空\*/**

……

node是一个指针变量，这个指针变量的地址在程序载入内存时就已经确定,并且不能修改，所以我们可以用 LinkList \*\*addr = &node获得它的地址，但node指向的是NULL，还没有分配内存。这里只是说明二级指针的用法，在实际编程中，我们一般用 LinkList \*create\_node();这样的原型（每个单位都有具体的程序规范）。这样使用就方便,例如：

**LinkList \*linklist = ……;**

**……**

**LinkList \*node = create\_node();**

**insert\_node(linklist,node );**

**……**

1. 函数的指针参数在方法体内的改变，不会影响到调用函数的程序。但指针指向的内容在函数的改变，会反映到调用函数的程序或方法。
2. C语言程序要求对指针的要求遵循“谁申请谁释放”的原则。即使申请的指针在数据结构内部，那么也要提供释放的方法供调用者使用。一个字节的泄露，在一次运行时没有什么问题，但在服务程序中，很有可能导致服务器宕机，这就是大家提的“7X24小时服务”的内涵。
3. C语言“申请地址要初始化，释放指针后要赋NULL”这是铁律。主要原因是误用释放后的指针或误用没有初始化的内容很快被找出。可以定义宏来简化。

**#define MALLOC(\_\_ptr,\_\_size) (void \*) \_\_ptr = (void \*)malloc(\_\_size); \**

**memset(\_\_ptr,0,\_\_size)**

**#defien FREE(\_\_ptr) free(\_\_ptr); \**

**\_\_ptr = NULL**

1. **链表的设计技巧**

**链表的设计，要看使用的目的，插入、删除、查找的特点进行设计。比喻有的链表要求记录有序的数据，我们可以增加一个尾部指针，简化插入操作，而且可以保证插入数据的先后。 链表是非常基础的，以后会经常用到。我们的计算机的文件系统物理存储都是分块存储的，每块之间链接在一起（有指针吗？）。下面举例说明：**



