笔记

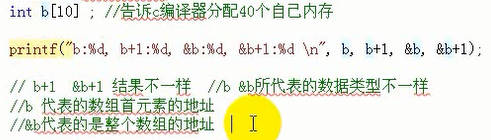
C语言提升

Day1

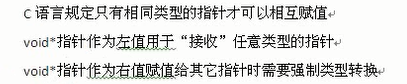
1：



2：



3：malloc返回的指针是void类型，因此用malloc创建的指针需要强制类型转换来赋给另一个指针



4：间接修改变量

int a;

Print(“&a:%d”,&a); //返回1245024

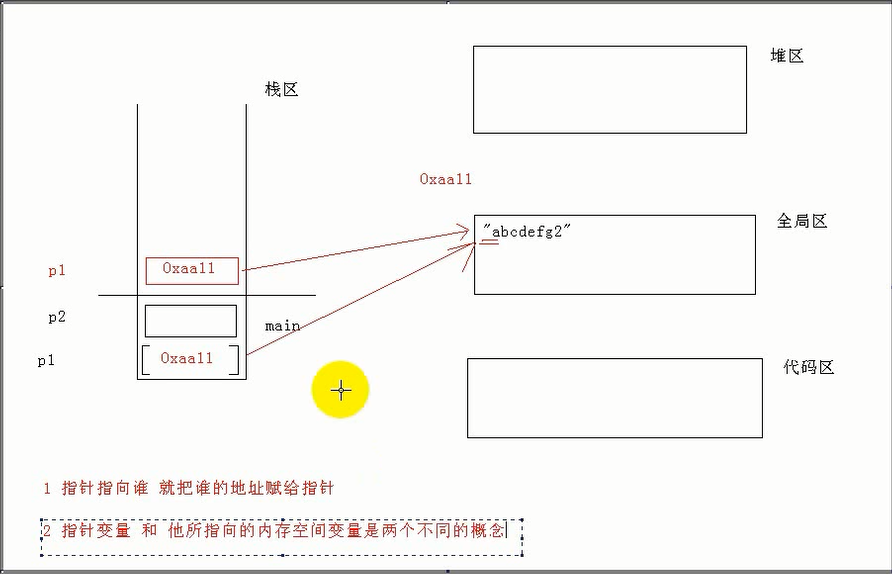
\*((int\*)1245024)=200; //间接修改变量（1）

int \*p;

p=1245024;

\*p=200; //间接修改变量（2）

5：



6：

C:\Users\LEWISR~1\AppData\Local\Temp\1518593109(1).png

C:\Users\LEWISR~1\AppData\Local\Temp\1518593246(1).png

7：

C:\Users\LEWISR~1\AppData\Local\Temp\1518593496(1).png

//strcpy把“123456789”copy在buf所指向的内存空间

//buf在栈区，“123456789”copy到栈区

C:\Users\LEWISR~1\AppData\Local\Temp\1518593597(1).png

//“abcdefg2”是在全局区

8：！！！

判断栈的生长方向

后进栈的值比先进栈的值大，则栈开口向上。

后进栈的值比先进栈的值小，则栈开口向下。

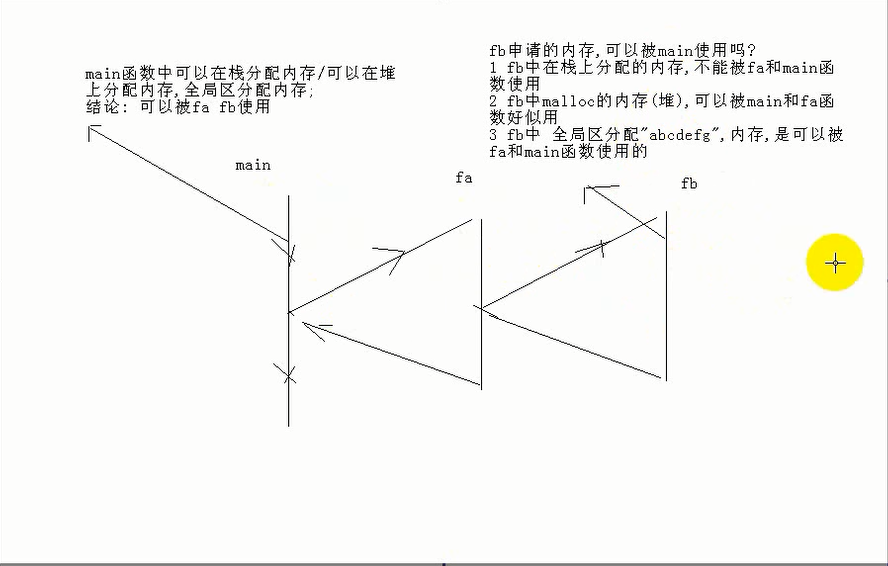
一般栈开口向下。

不管栈开口向上还是向下，数组buf进栈，buf+1永远向上。（因为buf+1的值比buf大）

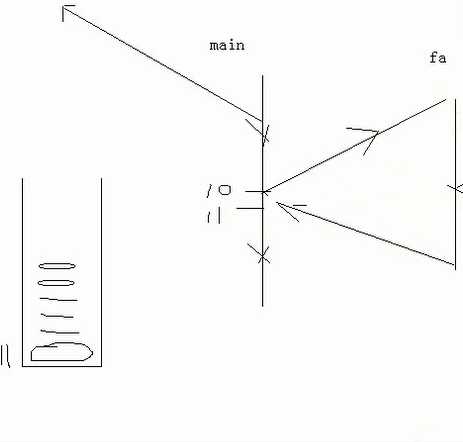
堆区的生长方向是向上的。

9：！！！

main函数内分配好内存以便函数使用



10：



main函数在第十行调用fa函数

栈区变化：

main第十一行先进栈

接着是fa的形参入栈，然后到fa的变量入栈

执行完fa，fa的变量出栈，然后到fa的形参出栈

再到main第十一行出栈。

11：！！！

C:\Users\LEWISR~1\AppData\Local\Temp\1518672087(1).png

12：！！！

C:\Users\LEWISR~1\AppData\Local\Temp\1518672400(1).png

\*p在=号左边都是把所指向内存块的首地址赋给指针。

\*p在=号右边都是把指针所指向的内存块的内容都取出来。

13：！！！

全局变量区不能被修改。

例子：

Void change(int a)

{  
a = 100; //这里出错，a已经指向了全局变量不可修改

}

void main()

{

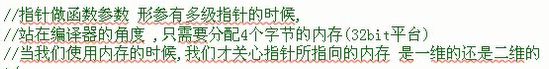
Void change(int a);

Int a = 10; //a指向全局变量

Change(a);

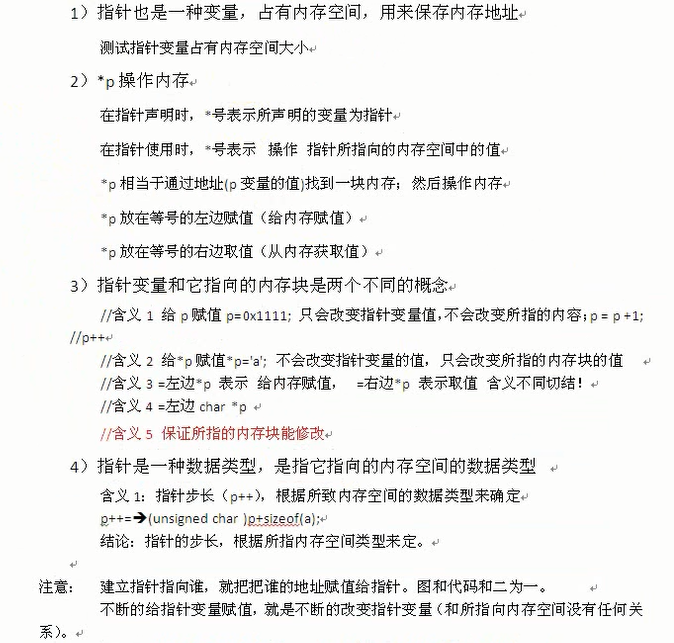
}

14：



15：

指针铁律1：



（分清楚图中“注意”的内容）

16：！！！

指针变量相当于一个标签，这个标签指向一块内存空间。

17：！！！

strcpy（p,”111222”）

把全局区“111222”内容复制到p所指向的内存块

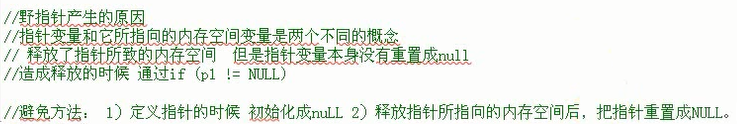
strcpy的本身属性：即strcpy只用于字符串复制，并且它不仅复制字符串内容之外，还会复制字符串的结束符

18:！！！

指针P用malloc赋值后

free（p）

p所指向堆区的内容删除，但是p任然指向已删除的堆区，此时p就成为野指针。



释放内存：把堆区内存空间free后

把指针变量赋值为NULL

19：

strcmp（p1，p2）

比较p1和p2所指向的内存空间的内容

20：!!!

char \*p = NULL;

strcpy（p，“abcdef“）

会发生错误，因为p指向NULL内存块，而NULL内存块是受系统保护无法修改的

strcpy则修改p所指向的内存块，这就会发生错误。

21：  
p1 = （char\*）malloc（100）；

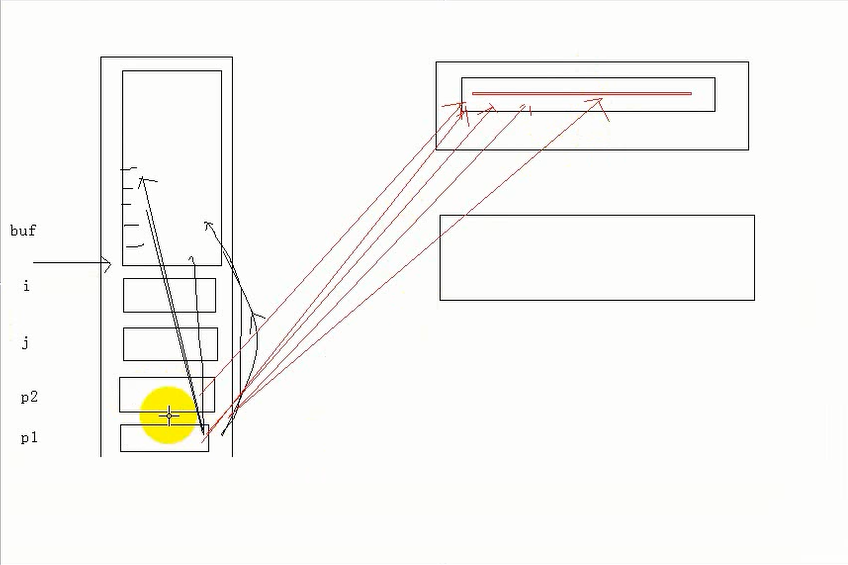
char \*p2 = p1； //把p1的首地址赋值给p2

for（int i=0;i<100;i++） //相当于逐个向后遍历

{

P2 = p1+1;

}



22：！！！

要实现运用函数间接修改变量的值，需要把变量的地址传给函数才能实现。

int getfilelen（int \*p）//这样才能实现

{

\*p = 40；

}

int getfilelen1（int b）//这样不能实现

{

b = 40；

}

int main（）

{

int a = 30；

a = getfilelen（&a）；

print（“a：%d“，a）；//这样才能实现

a = getfilelen1（a）

print（“a:%d”,a）;//这样不能实现

23：！！！

int getfilelen（int b）

{

b = 100；

}

这个函数和

int getfilelen1（）

{

int b；

b = 100；

}

函数的功能一样只不过getfilelen可以把实参传递给形参，有了一个对外的属性，其它和getfilelen1的功能一样

24：！！！

类比22

int getfilelen（int \*\*p）//这样才能实现

{

\*p = 40；

}

int getfilelen1（int \*p）//这样不能实现

{

p = 40；

}

int main（）

{

int \*p = 30；

p = getfilelen（&p）；

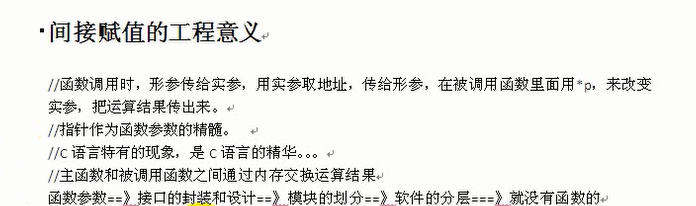
print（“p：%d“，p）；//这样才能实现

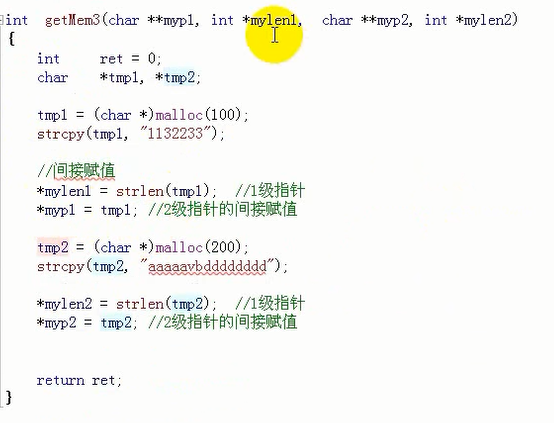
p = getfilelen1（p）

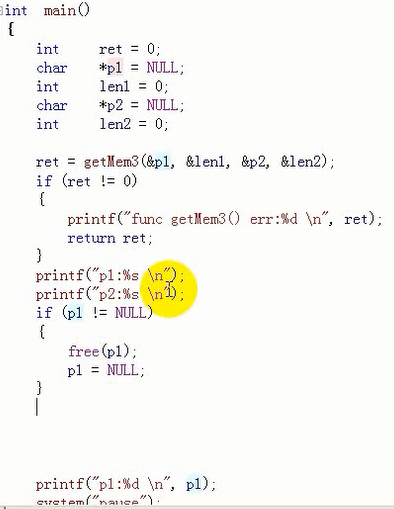
print（“p:%d”,p）;//这样不能实现

22-24：

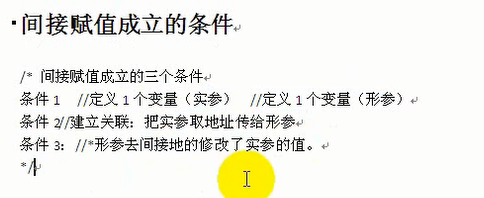
指针做函数参数的重要性！！！



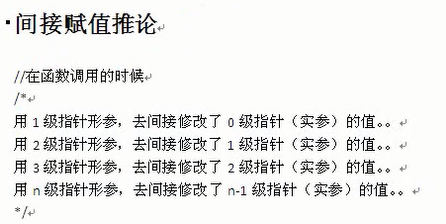




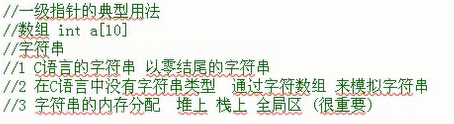
25：



26：



27：



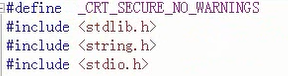
28：

快捷输入

Vs2017 ->

Vassistx ->

设置快捷键#1 ->

输入 ->

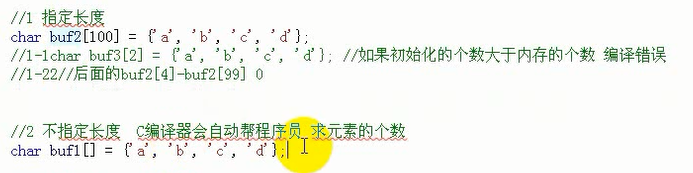
29:！！！

函数return只能返回一个值

函数以指针传地址的方式修改内存空间可以输出几个值

30：

字符串数组1



31：

字符串数组2

Strlen（）求字符串的大小。

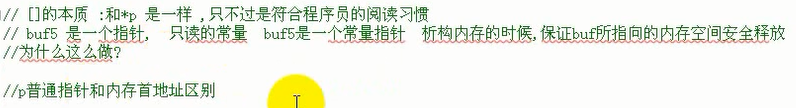
Sizeof（）求字符串数组的大小。



32：



33：！！！



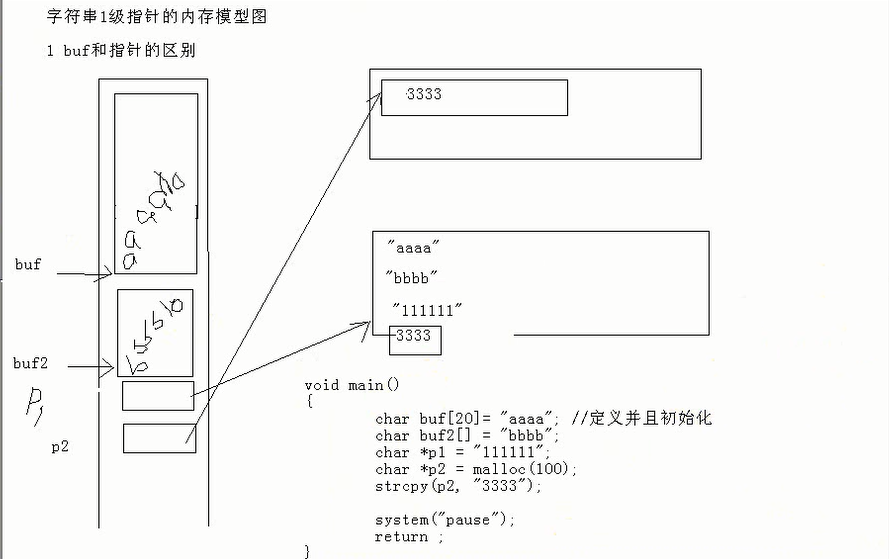
Char Buf[128];

Buf = Buf+1；//是不可以的

而p=buf；

P=p+1；//这样就可以

34：



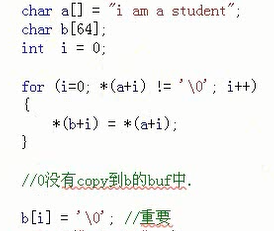
35：！！！

字符串数组buf[128]中一共有129个元素，最后一个是’\0’；

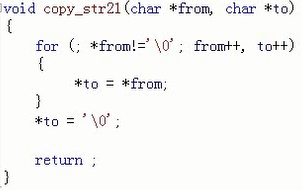
如果不把字符串数组的最后一个元素定义为’\0’，那么打印字符串数组是就会出现“烫烫烫烫烫烫烫”

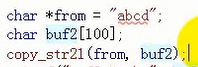
36：

字符串数组拷贝的普通做法：



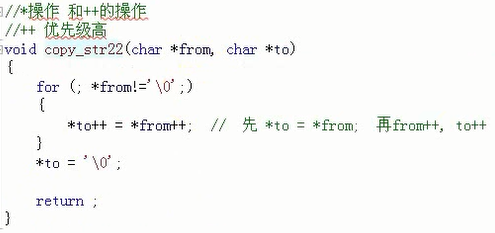
字符串数组拷贝的优化方法1：



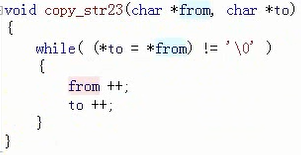


及其内存示意图：

字符串数组拷贝的优化方法2：（to++表示先读取to然后再执行++）

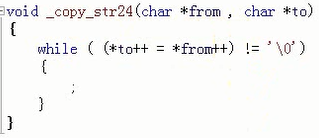


字符串数组拷贝的优化方法3：

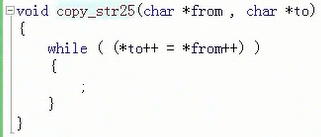


先执行\*to=\*from后判断是否不等于’\0’

字符串数组拷贝的优化方法4：



字符串数组拷贝的优化方法5：



37：

函数api

例如：void copy\_str(char \*in /\* in \*/ , char \*out /\* out \*/ )

In是仅仅传入函数形参和实参的类型一样

Out是传入函数并进行函数内修改，那么形参必须指向实参的内存块

例如：

实参：char a ， 形参：char \*a ， 函数：void formula（char \*a）， 传入方式void formula（&a）

实参：char \*a ， 形参：char \*\*a ， 函数：void formula（char \*\*a）， 传入方式void formula（&a）

实参：char \*\*a ， 形参：char \*\*\*a ， 函数：void formula（char \*\*\*a）， 传入方式void formula（&a）

以此类推……

38：！！！

不要随意改变形参的值（十分重要）

例如：



跳出while循环后from指向’\0’的内存空间

只能打印’\0’

修改方法：

引入tmpfrom，tmpto辅助指针



39：

Strstr函数：

Strstr(p,”abcd”); //查找p中的“abcd”，查找到就返回“abcd”开头的地址，查找不到返回NULL

40：

\*p++; //++的优先级高于\*

41：  
（1）printf(“func FunctionName() err:%d(The warning part)\n”); //提示出错的地方方便找错

（2）函数调用前先判断传入的实参是否为空，若为空返回报错内容

42：

Char \*p = NULL;

Char \*p = 0;

两个指针指向同一个位置。

43：

Strlen和sizeof的区别



44：

Isspace函数

用法：#include <ctype.h>

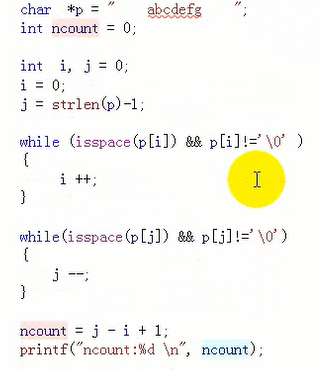
功能：判断字符c是否为空白符

说明：当c为空白符时，返回非零值，否则返回零。

空白符指空格、水平制表、垂直制表、换页、回车和换行符。

45：

用isspace函数消去字符串两边的等号



45：

Strcpy和strncpy的区别

strcpy只是复制字符串，但不限制复制的数量，很容易造成缓冲溢出

strncpy可以控制复制字符串的长度。

用法例子：

char name[]="Chinane";

dest[20]={0};

strncpy(dest,name,9); //name为要拷贝的字符串的开头

记得最后的是’\0’

46：

Char \*p = “ abcdefg “; //指针指向全局变量不可以修改，因此不能执行p++

Char buf[1024] = “ abcdefg “; //创建buf数组在栈区buf指向全局变量区，但是buf可以实现buf++，因为buf为数组

47：！！！

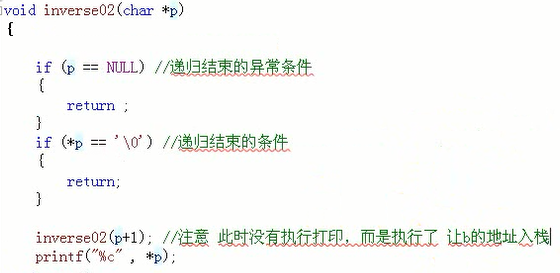
字符串反转

方法一： //两头堵模型



方法二：

（函数调用，利用堆栈的先进后出的特性实现字符串反转，要注意递归结束否则会导致堆栈溢出）请看函数递归，（笔记52）



方法二推广：



48：

函数调用（递归）

要注意递归结束条件，以防堆栈溢出

49：

Memset函数功能

例子：

char \*s = “abcdefg“；char c = ‘0’；int ncount = 3;

Memset（s ， c ， ncount）；

Printf(“%s”,s);

输出000defg

50：

Strcat函数的功能

例子：

将两个char类型连接。

char d[20]="GoldenGlobal"; char \*s="View";

strcat(d,s);

结果放在d中

printf（"%s",d）；

输出 d 为 GoldenGlobalView （中间无空格）

d和s所指内存区域不可以重叠且d必须有足够的空间来容纳s的字符串。

返回指向d的指针。

51：

Strncat（和strncpy类比）

Strncat可以控制连接字符串的长度

例子：

char d[20]="GoldenGlobal"; char \*s="View";

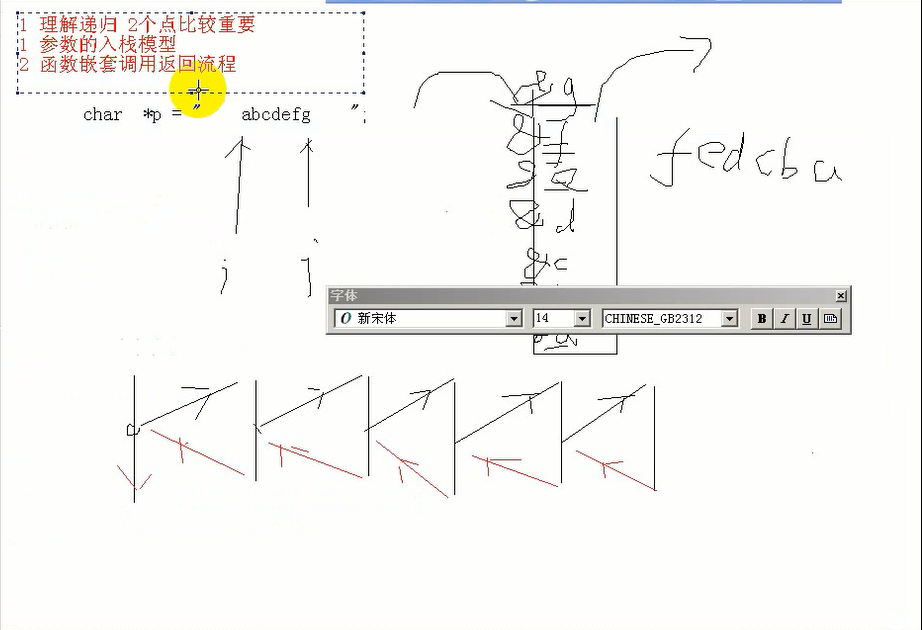
strncat(d , s , 3);

printf(%s”,d);

输出d为GoldenGlobalVie

52：！！！

函数递归模型：



53：

Char \*p = {0}; char \*p = 0; char \*p = NULL; 同一个含义。

54：！！！

1.防止越界

（1）Char buf[3] = “ab”; //记得’\0’也占一个字节

（2）char \*a = （char \*）malloc(sizeof(50)); a++; free(a); //此时会出错，因为free的是a的首地址，而a++已经改变了a的地址，因此出错。

2.不要随便改变形参的值

要新建一个辅助变量

3.要在主函数分配好内存再在子函数引用

4.\*p++和\*(p++)含义一样（千万别理解成和（\*p）++一样）

55：

二级指针所指向的内存空间赋值给一级指针所指向的内存空间

//a为一级指针，b为二级指针

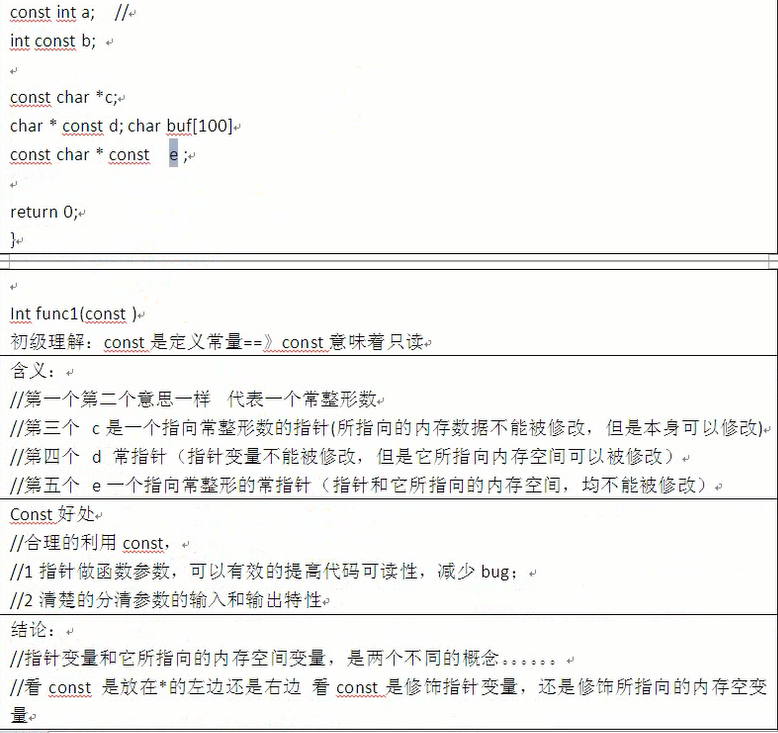
例子：

\*a = \*（\*b））；

\*a++=\*（\*b）++；

其他级以此类推。

56：！！！(注意看图)



解释：

Const（常整形数，不可以修改）

Const int a;

Int const b;

A,b都是代表常整形数

Const char \*c；

Char \* const d；

Const char \* const e；

C代表c指针指向的内存空间是常整形数

D代表d指针变量是常整形数

E代表e指针变量是常整形数，e指针所指向的内存空间也是常整形数

57：

在c语言中 //const在c语言中形同虚设

Const int a = 10;

Int \*p = &a;

\*P = 100;

Printf(“%d”,a);

在c++中 //const在c++中发挥作用

58：

求数组长度：

Int array{}={…………};

Int num = sizeof(array)/sizeof(array[0]);

59：！！！

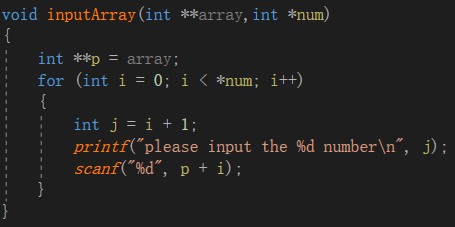
Int \*array[]={…………};

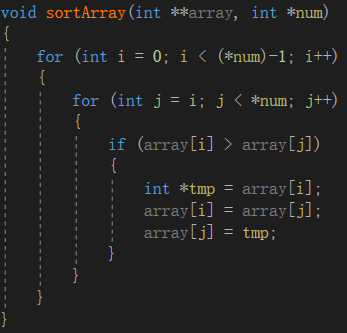
Int \*tmp = array[i]; //交换的只是指针的值即数组的元素（改变的是指针的指向，并没有改变内存空间）

60：！！！二维数组第一种内存模型

Int \*array[]={…………};

//二级指针的输入

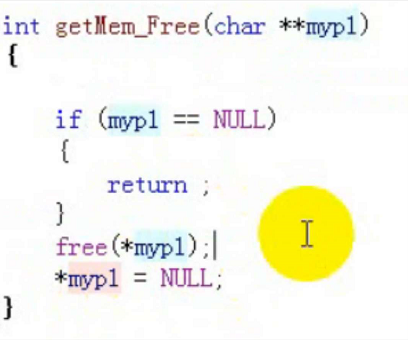






61：

用malloc创建的二级指针的释放方法



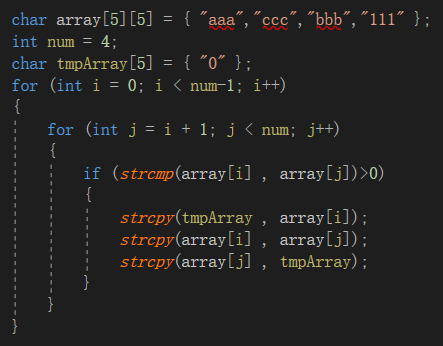
62：！！！



指针做输出：函数内定义的指针变量比函数形参第一级！！！最后通过\*形参=函数内定义指针，把结果甩出去。

63：！！！

Int array[i][j] = {0}; //定义i行j列的数组



字符串数组对比要用strcmp函数

字符串拷贝不弄用=，要用strcpy函数

上图strcpy(tmpArray , array[i]);

Strcpy(array[i] , array[j]);

Strcpy(array[j] , tmpArray);

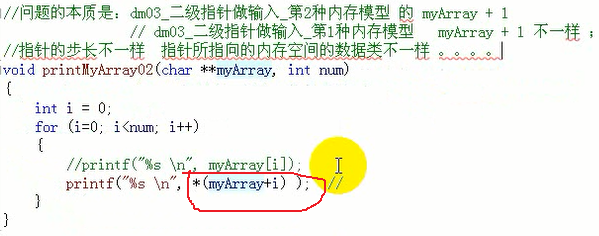
实质交换的是内存块。

64：！！！

Int \*array[i] = {0};

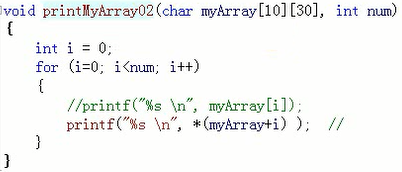
Int array[i][j] = {0};

上面两个定义的二级指针的数组步长不一样



因为步长问题，上面红圈部分会导致数组越界

改正方法：



问题所在：

函数形参的定义影响函数对数组步长的定义

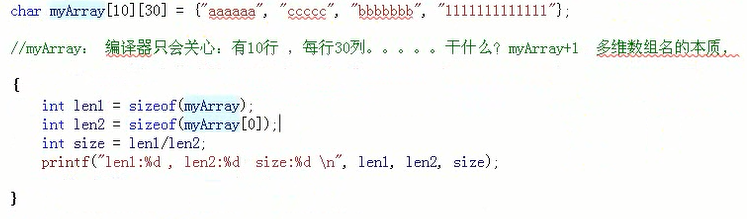
以下为正确的形参和数组的对应：

形参：\*\*array，数组：\*array[i]

形参：array[i][j]，数组：array[i][j]

下面会有更多解决办法

65：！！！



输出结果len1：300，len2：30，size：10

Sizeof(myArray[0])求的是数组每一行有几个元素

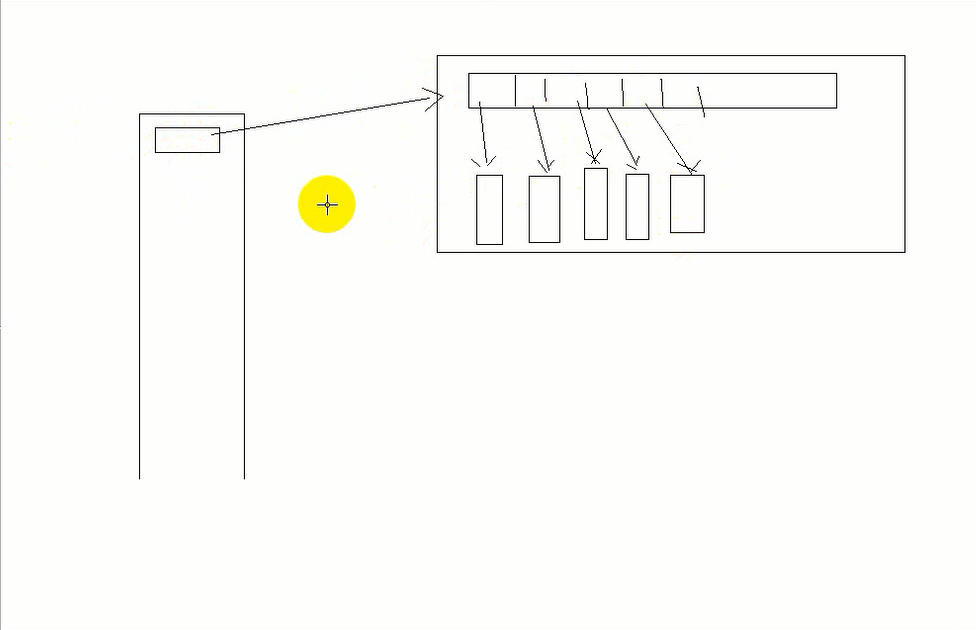
二维数组MyArray+1 //一次跳30个元素

66：

Sprintf函数格式：

Sprintf（要写入的地址（如array）,”写入格式（如%d）”,写入内容(如num)）

67：！！！二维数组第三种内存模型（array[i][j]的实质）



栈区：

Char \*\*p2=NULL；

堆区：

P2 = (char \*\*)malloc(sizeof(char \*)\*num); //num已经定义好

For(int i = 0;i<num;i++)

{

P2[i]=(char \*)malloc(sizeof(char)\*100);

Sprintf(p2[i],”%d%d%d”,i+1,i+1,i+1);

}

打印二维数组：

For(int I = 0;i<num;i++)

{

Printf(“%s”,p2[i]);

}

释放堆区内存：

For(int I =0;i<num;i++)

{  
if(p2[i]!=NULL)

{

Free (p2[i]);

P2[i]=NULL;

}

}

If(p2!=NULL)

{

Free(p2);

}

排序方法一： //交换指针所指向的内存空间

For（int I = 0;i<num;i++）

{

For(int j= i+1;j<num;j++)

{

If(strcmp(p2[i],p2[j])>0)

{

Char \*tmp = p2[i];

P2[i]=p2[j];

P2[j]=tmp;

}

}

}

排序方法二： //改变内存空间数据

For（int I = 0;i<num;i++）

{

For(int j= i+1;j<num;j++)

{

If(strcmp(p2[i],p2[j])>0)

{

Strcpy(char tmp[100] , p2[i]);

Strcpy(P2[i] , p2[j]);

Strcpy(P2[j] , tmp);

}

}

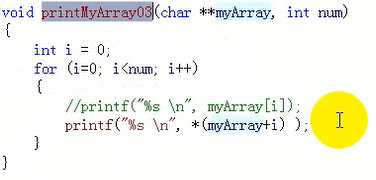
}

函数封装实现：

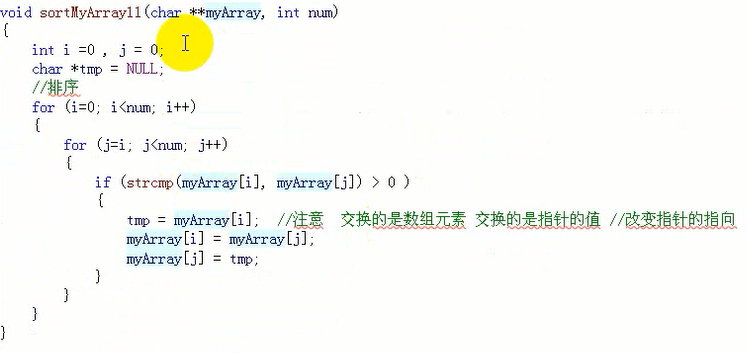
创建二维数组：



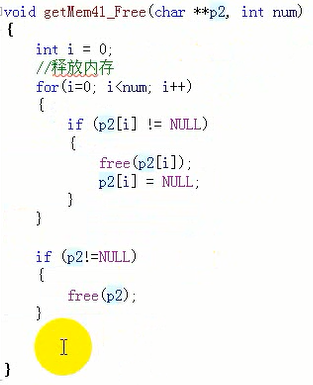
打印数组：



数组排序：

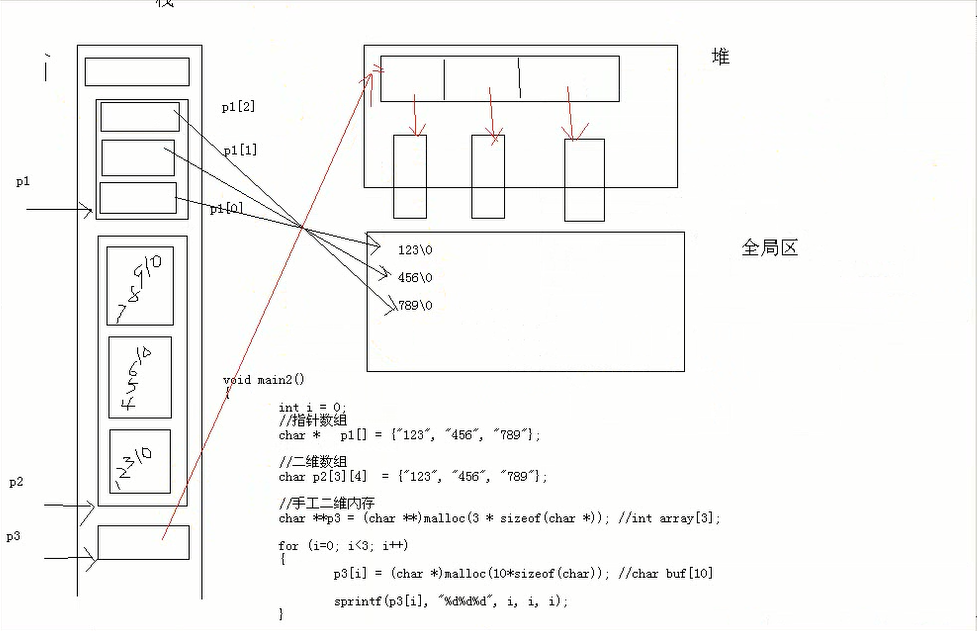


释放堆区：



（此处p2释放后变成野指针）

68：！！！



注意：

array[3][n]={“123”,”456”,”789”}; 注意：array[i][j]的步长为j

123,456,789是直接填进数组的不需要指向全局区

注意：

Char \*array[ ] 和char \*\*p2 = (char \*\*)malloc(3\*sizeof(char\*)) 的步长都是4

69：！！！

例子：函数封装：三级指针形参修改二级指针实参：

基本理念：

void getMem(char \*\*\* p3 , int num)

{

Char \*\*tmp = NULL; //辅助变量，不要轻易改变形参

if(p3 =NULL) //千万不要写成\*p3 = NULL，因为\*p3可以为NULL

{

return;

}

………….

………….

………….

\*p3 = tmp; //把辅助指针接回来

}

在主函数中打印：

Int main()

{

Char \*\*p2 = NULL;

void getMem(char \*\*\* p3 , int num);

getMem(&p2 , num);

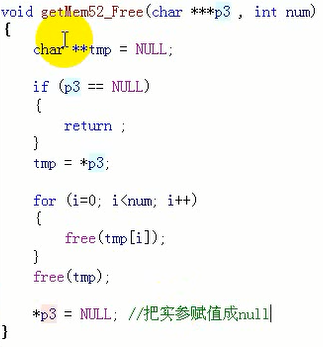
for(int I = 0;i<num;i++)

{

printf(“%s”,p2[i]); //假设p2为数组

}

函数封装：释放二级指针



注意：不要轻易改变形参的值

释放堆区内存后，把指针赋值为NULL，避免野指针

70：

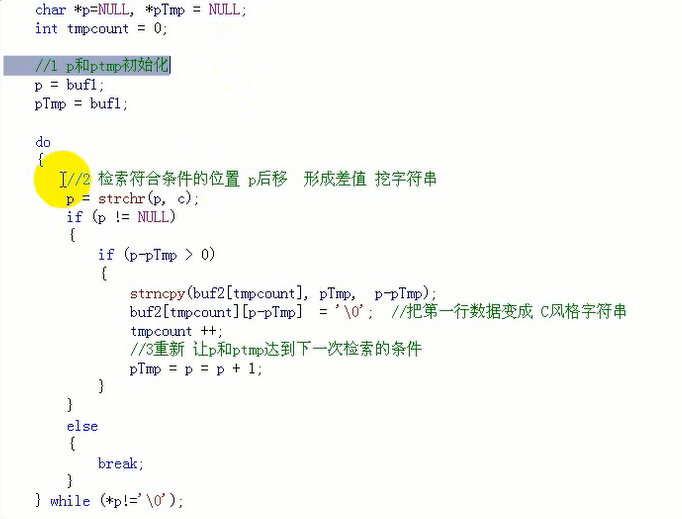
Strchr函数例子：

Strchr（string，“.”）; //即在字符串string中找出字符”.”的首次出现位置的地址

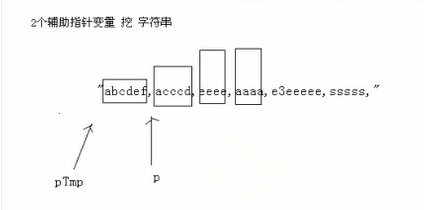
对比strstr函数，strstr函数时查找字符串中另一个字符串首次出现的位置

71：

字符串按指定字符分开拷贝：



图解：



72：

Malloc后的内存不释放成为内存泄漏。

73：！！！

如71题目：

按照有多少个“，”分配多少个行，再用strncpy分配多少个列

74：！！！

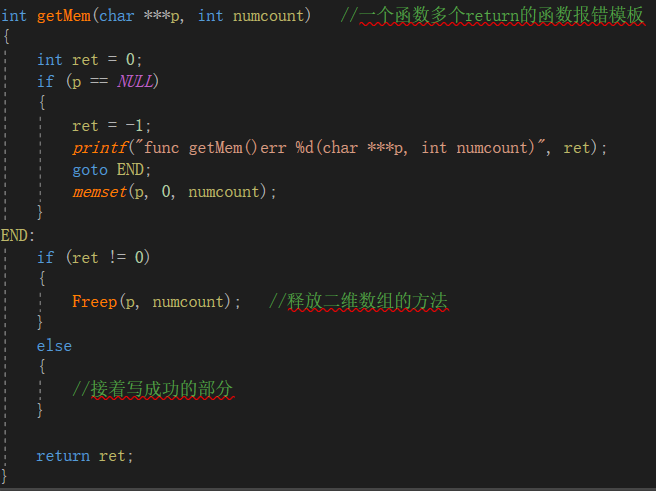
（切记看day4的第二个视频，本笔记内容稍不规范）

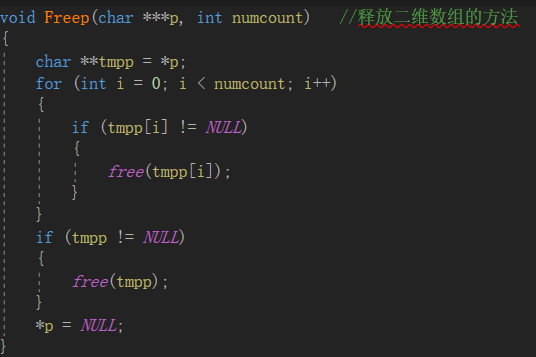
在函数出错的时候要有报错提示，

并且要将前面申请的内存全部释放掉。

例子：

专门针对一个函数多个return的函数报错模板（并且释放之前申请的内存！！！）





75：