都哥 2020/11/20 21:50:09  
int a[3][4] = {0};  
printf("%d\n",sizeof(a)); //48  3\*4\*4 相当于数组的大小  
printf("%d\n",sizeof(a[0][0])); //4 相当于第一个元素占用字节的数量  
printf("%d\n",sizeof(a[0])); //16 a[0]相当于第一行作为1维数组的数组名，所以计算的是第一行的大小  
printf("%d\n",sizeof(a[0]+1)); //4 a[0]没有单独放在了sizeof，所以不能认为是第一行作为1维数组的数组名，所以代表的是第一行的数组的首地址，加1，就相当于往后偏移一个位置，也就是第一行第二个元素的地址  
可以打印printf("%p\n", &a[0][0]) 和 printf("%p\n", a[0]+1), 地址大小是4字节  
  
printf("%d\n",sizeof(\*(a[0]+1))); //4 相当于第一行第二列的元素大小是4字节  
printf("%d\n",sizeof(a+1)); //4 a代表二维数组首行的地址，加1表二维数组第二行的地址  
  
都哥 2020/11/20 21:49:53  
1.先结合图解， 说明p是一个指针变量保存的是静态字符串"abcdef"的地址  
char \*p = "abcdef";  
printf("%d\n", sizeof(p)); //4 计算指针变量p的大小  
printf("%d\n", sizeof(p+1)); //4 p+1 计算的是b的地址， 地址的大小也是4  
printf("%d\n", sizeof(\*p)); //1 拿到的是字符‘a’， a占用1字节  
printf("%d\n", sizeof(p[0]));//1 p[0] 就相当于 \*(p+0) 就相当于字符 ‘a’  
printf("%d\n", sizeof(&p));//4 p的地址  
printf("%d\n", sizeof(&p+1));//4 同样还是地址  
printf("%d\n", sizeof(&p[0]+1));//4 相当于字符‘b’的地址  
  
printf("%d\n", strlen(p)); //6  
printf("%d\n", strlen(p+1)); //5  
printf("%d\n", strlen(\*p)); //err  
printf("%d\n", strlen(p[0])); //err  
printf("%d\n", strlen(&p)); // 随机值，因为p当中保存的是“abcdef”的地址， 地址的构成是没有办法预测的，并且是否有\0也是说不清楚的，有可能越界，也有可能越界不了。画图最清晰  
  
printf("%d\n", strlen(&p+1));// 随机值越界了，已经越过了p保存的地址，到后一个字节，本质已经越界，并且到底啥时候预见\0那是没有办法预测的，所以，不崩溃的前提下，就是随机值  
  
printf("%d\n", strlen(&p[0]+1)); //5  
  
都哥 2020/11/20 21:49:35  
char arr[] = "abcdef";  
printf("%d\n", sizeof(arr)); //7 计算整个字符数组的大小，包含\0  
printf("%d\n", sizeof(arr+0)); //4 计算的是首元素的地址  
printf("%d\n", sizeof(\*arr)); //1 计算的是第一个元素的大小，也就是‘a’  
printf("%d\n", sizeof(arr[1])); //1 计算第二个元素的大小， 也就是‘b’  
printf("%d\n", sizeof(&arr)); //4 计算数组的地址， 本质还是地址，所以，在32位平台地址为4字节，在64位平台地址为8字节  
printf("%d\n", sizeof(&arr+1)); //4  本质上还是地址， 所以，在32位平台地址为4字节，在64位平台地址为8字节  
printf("%d\n", sizeof(&arr[0]+1)); //4 本质上还是地址， 第二个元素：‘b’的地址，所以，在32位平台地址为4字节，在64位平台地址为8字节  
  
printf("%d\n", strlen(arr)); //6 字符长度  
printf("%d\n", strlen(arr+0)); //6 首元素的地址+0, 并没有向后偏移，所以还是6  
printf("%d\n", strlen(\*arr)); //代码错误  
printf("%d\n", strlen(arr[1])); //代码错误  
printf("%d\n", strlen(&arr)); //6字符， 取得是数组的地址， 本身数组地址应该传递给数值指针， 但是现在传递给strlen，接收一个字符指针的函数，就是报警告，间接级别不同  
  
printf("%d\n", strlen(&arr+1)); //代码错误, 已经越过了整个数组，所以内存访问越界了，有可能会崩溃  
并且传递的参数类型也不同，会报警告  
printf("%d\n", strlen(&arr[0]+1)); //5  从‘b’开始算了，直到\0一种是5个字符  
  
都哥 2020/11/20 21:49:11  
char arr[] = {&apos;a&apos;,&apos;b&apos;,&apos;c&apos;,&apos;d&apos;,&apos;e&apos;,&apos;f&apos;};  
printf("%d\n", strlen(arr)); //随机值，字符数组中没有\0, 所以不知道什么时候结束  
printf("%d\n", strlen(arr+0));//随机值，相当于strlen(arr)  
printf("%d\n", strlen(\*arr));//代码报错， strlen的参数接收的是一个地址，当我们使用\*arr传参的时候，相当于将首元素的值传递给了strlen， 也就是‘a’, strlen是认为传递进来的a，也是也是一个地址， a的ascii码值为97,所以，strlen会从97后面去找，遇见\0就会停止，但是这个做法导致了越界的风险！  
printf("%d\n", strlen(arr[1]));//代码报错， 同上  
printf("%d\n", strlen(&arr));//随机值  
printf("%d\n", strlen(&arr+1));//随机值-6，但是已经越界访问了，有可能导致代码崩溃， 因为&arr+1，已经将整个数组都偏移过去  
printf("%d\n", strlen(&arr[0]+1));//随机值-1  
  
都哥 2020/11/20 21:48:54  
int a[] = {1,2,3,4};  
printf("%d\n",sizeof(a));  // 16 sizeof(数组名) 计算数组总占用字节的数量  
  
printf("%d\n",sizeof(a+0)); //4 a表示首元素的地址， a+0，地址没有变化，在32位平台地址为4字节，在64位平台地址为8字节  
  
printf("%d\n",sizeof(\*a)); //4 a表示首元素地址， 解引用拿到首元素的值， 类型是int， 所以为4字节  
  
printf("%d\n",sizeof(a+1)); //4 a表示首元素地址，a+1是第二个元素的地址。在32位平台地址为4字节，在64位平台地址为8字节  
  
printf("%d\n",sizeof(a[1])); //4 a[1] 表示访问数组的第二个元素， 类型为int，所以占用4字节  
  
printf("%d\n",sizeof(&a));   //4 &a取出的是数组的地址，本质还是地址，在32位平台地址为4字节，在64位平台地址为8字节  
  
printf("%d\n",sizeof(\*&a));  // 16 &a数数组的地址， 数组的地址解引用访问的是整个数组， sizeof计算的也就是整个数组的大小  
  
printf("%d\n",sizeof(&a+1)); //4 &a取出的是数组的地址，+1，偏移整个数组的大小，但是本质还是地址，所以，在32位平台地址为4字节，在64位平台地址为8字节  
  
printf("%d\n",sizeof(&a[0])); //4 取的是第一个元素的地址，所以，在32位平台地址为4字节，在64位平台地址为8字节  
  
printf("%d\n",sizeof(&a[0]+1)); //4 取得是第二个元素的地址，本质还是地址，所以，在32位平台地址为4字节，在64位平台地址为8字节  
  
数组名是数组首元素的地址，但是有两个例外：  
1.sizeof(数组名)  --》 这里的数组名表示整个数组  
2.&数组名 --》数组名表示整个数组  
  
  
  
//字符数组  
char arr[] = { &apos;a&apos;, &apos;b&apos;, &apos;c&apos;, &apos;d&apos;, &apos;e&apos;, &apos;f&apos; };  
printf("%d\n", sizeof(arr)); // 6  
printf("%d\n", sizeof(arr + 0)); //4  
printf("%d\n", sizeof(\*arr)); //1  
printf("%d\n", sizeof(arr[1])); //1  
printf("%d\n", sizeof(&arr)); //4  
printf("%d\n", sizeof(&arr + 1)); //4  
printf("%d\n", sizeof(&arr[0] + 1)); //4