<https://blog.csdn.net/huqinwei987/article/details/23597091>

有些基础知识快淡忘了，所以有必要复习一遍，在不借助课本死知识的前提下做些推理判断，温故知新。

**1.联合体union的基本特性——和struct的同与不同**

union，中文名“联合体、共用体”，在某种程度上类似结构体struct的一种数据结构，共用体(union)和结构体(struct)同样可以包含很多种数据类型和变量。

不过区别也挺明显：

结构体(struct)中所有变量是“共存”的——优点是“有容乃大”，全面；缺点是struct内存空间的分配是粗放的，不管用不用，全分配。

而联合体(union)中是各变量是“互斥”的——缺点就是不够“包容”；但优点是内存使用更为精细灵活，也节省了内存空间。

**2.双刃剑——多种访问内存途径共存**

一个例子了然：

1. *//example*
2. #include<stdio.h>
3. union var{
4. long int l;
5. int i;
6. };
7. main(){
8. union var v;
9. v.l = 5;
10. printf("v.l is %d\n",v.i);
11. v.i = 6;
12. printf("now v.l is %ld! the address is %p\n",v.l,&v.l);
13. printf("now v.i is %d! the address is %p\n",v.i,&v.i);
14. }
15. 结果：
16. v.l is 5
17. now v.l is 6! the address is 0xbfad1e2c
18. now v.i is 6! the address is 0xbfad1e2c

所以说，管union的叫共用体还真是贴切——完全就是共用一个内存首地址，并且各种变量名都可以同时使用，操作也是共同生效。如此多的access内存手段，确实好用，不过这些“手段”之间却没法互相屏蔽——就好像数组+下标和指针+偏移一样。

上例中我改了v.i的值，结果v.l也能读取，那么也许我还以为v.l是我想要的值呢，因为上边提到了union的内存首地址肯定是相同的，那么还有一种情况和上边类似：

一个int数组变量a，一个long int(32位机中，long int占4字节，与int相同)变量b，我即使没给int变量b赋值，因为数据类型相同，我使用int变量b也完全会拿出int数组a中的a[0]来，一些时候一不小心用上，还以为用的就是变量b呢~

这种逻辑上的错误是很难找出来的（只有当数据类型相去甚远的时候稍好，出个乱码什么的很容易发现错误）。

**PS:感谢热心网友的提醒“在union定义结束时加分号”，其实是可以不加的，因为他不在主函数内，不是执行的语句，如果是主函数内声明的union就必须加分号了，在主函数内不加分号就涉及到基础常识了——没有分号隔开怎能叫一句。**

**3.联合体union和大小端（big-endian、little-endian）：**  
下边示范了一种用途，代表四个含义的四个变量，但是可以用一个int来操作，直接int赋值，无论内存访问（指针大小的整数倍，访问才有效率），还是时间复杂度（一次和四次的区别，而且这四次有三次都是不整齐的地址），都会低一些。

1. #include<stdio.h>
2. union var{
3. char c[4];
4. int i;
5. };
7. int main(){
8. union var data;
9. data.c[0] = 0x04;*//因为是char类型，数字不要太大，算算ascii的范围~*
10. data.c[1] = 0x03;*//写成16进制为了方便直接打印内存中的值对比*
11. data.c[2] = 0x02;
12. data.c[3] = 0x11;
13. *//数组中下标低的，地址也低，按地址从低到高，内存内容依次为：04,03,02,11。总共四字节！*
14. *//而把四个字节作为一个整体（不分类型，直接打印十六进制），应该从内存高地址到低地址看，0x11020304，低位04放在低地址上。*
15. printf("%x\n",data.i);
16. }

结果：  
11020304  
证明我的32位linux是小端（little-endian）

**4.联合体union所占内存空间大小：**  
  
前边说了，首先，union的首地址是固定的，那么，union到底总共有多大？根据一些小常识，做个不严谨不高深的基础版验证吧。  
  
根据：分配栈空间的时候内存地址基本上是连续的，至少同类型能保证在一起，连续就说明，我如果弄三个结构体出来，他们三个地址应该连着，看一下三个地址的间隔就知道了。

1. #include<stdio.h>
2. union sizeTest{
3. int a;
4. double b;
5. };
6. main(){
7. union sizeTest unionA;
8. union sizeTest unionB;
9. union sizeTest unionC;
11. printf("the initial address of unionA is %p\n",&unionA);
12. printf("the initial address of unionB is %p\n",&unionB);
13. printf("the initial address of unionC is %p\n",&unionC);
14. }

打印，可以看到结果：  
  
the initial address of unionA is 0xbf9b8df8  
the initial address of unionB is 0xbf9b8e00  
the initial address of unionC is 0xbf9b8e08  
  
很容易看出，8,0,8，这间隔是8字节，按double走的。  
  
怕不保险，再改一下，把int改成数组，其他不变：

1. union sizeTest{
2. int a[10];
3. double b;
4. };

打印  
  
the initial address of unionA is 0xbfbb7738  
the initial address of unionB is 0xbfbb7760  
the initial address of unionC is 0xbfbb7788  
  
88-60=28  
60-38=28  
算错了？我说的可是16进制0x。那么0x28就是40个字节，正好是数组a的大小。

忘了提一个功能——**sizeof()**

用sizeof直接看，就知道union的大小了

1. printf("the sizeof   of unionA is %d\n",sizeof(unionA));
2. printf("the sizeof   of unionB is %d\n",sizeof(unionB));
3. printf("the sizeof   of unionC is %d\n",sizeof(unionC));
4. printf("the sizeof of union is %d\n",sizeof(union sizeTest));

**上边说的地址规律，没有特定规则，也可能和你的编译器有关。另外，那只是栈空间，还可以主动申请堆空间，当然，堆空间就没有连续不连续一说了。**

**5.联合体union适用场合：**  
  
有了前边那个验证，基本可以确认，union的内存是照着里边占地儿最大的那个变量分的。  
  
也就可以大胆的推测一下，这种union的使用场合，是各数据类型各变量占用空间差不多并且对各变量同时使用要求不高的场合（单从内存使用上，我觉得没错）。  
  
像上边做的第二个测试，一个数组（或者更大的数组int a[100]），和一个或者几个小变量写在一个union里，实在没什么必要，节省的空间太有限了，还增加了一些风险（最少有前边提到的逻辑上的风险）。所以，从内存占用分析，这种情况不如直接struct。

不过话说回来，某些情况下虽然不是很节约内存空间，但是union的复用性优势依然存在啊，比如方便多命名，这种“二义性”，从某些方面也可能是优势。这种方法还有个好处，就是某些寄存器或通道大小有限制的情况下，可以分多次搬运。

**6.本质&进阶：**

根据**union固定首地址**和**union按最大需求开辟一段内存空间**两个特征，可以发现，所有表面的定义都是虚的，所谓联合体union，就是在内存给你划了一个足够用的空间，至于你怎么玩~它不管~！（何止是union和struct，C不就是玩地址么，所以使用C灵活，也容易犯错）

没错，union的成员变量是相当于开辟了几个访问途径（即union包含的变量）！但是，没开辟的访问方式就不能用了？当然也能用！

写个小测试：

1. #include<stdio.h>
2. union u{
3. int i;
4. double d;*//这个union有8字节大小*
5. };
6. main(){
7. union u uu;
8. uu.i = 10;
9. printf("%d\n",uu.i);
11. char \* c;
12. c = (char \*)&uu;*//把union的首地址赋值、强转成char类型*
13. c[0] = 'a';
14. c[1] = 'b';
15. c[2] = 'c';
16. c[3] = '\0';
17. c[4] = 'd';
18. c[5] = 'e';
19. *//最多能到c[7]*
20. printf("%s\n",c);*//利用结束符'\0'打印字符串"abc"*
21. printf("%c %c %c %c %c %c\n",c[0],c[1],c[2],c[3],c[4],c[5]);
22. }

一个例子了然，我的结构体只定义了int和double“接口”，只要我获得地址，往里边扔什么数据谁管得到？这就是C语言（不止union）的本质——只管开辟一段空间。

**但是你获取地址并访问和存取的数据，最好确定是合法（语法）合理（用途符合）的地址，不然虽然能操作，后患无穷，C的头疼之处，可能出了问题你都找不到。**

====================================================================================================================================

# 补充：

## 补充1：

解决一下捧场网友的困惑。

关于“有名”与“无名”联合体在结构体内所占空间的问题，其实这和是不是结构体无关，只和“有名”、“无名”有关，而且有名无名也是表象，其实是声明类型与定义变量的区别，看例子，直接打印，

1. #include <stdio.h>
2. struct s1{
3. union u{
4. int i;
5. };
6. struct ss1{
7. int i;
8. };
9. };
11. struct s2{
12. union{
13. int i;
14. };
15. struct{
16. int i;
17. };
18. };
20. struct s3{*//the same to s2*
21. union su3{
22. int i;
23. }su33;
24. struct ss3{
25. int i;
26. }ss33;
27. };
29. union su4{
30. int i;
31. };
32. struct ss4{
33. int i;
34. };
35. struct s4{*//the same to s3*
36. union su4 su44;
37. struct ss4 ss44;
38. };
39. struct s5{*//the same to s1*
40. union su4;
41. struct ss4;
42. };
44. struct s6{*//the same to s1*
45. union{
46. int;
47. };
48. struct{
49. int;
50. };
51. };
53. main(){
54. struct s1 sVal1;
55. struct s2 sVal2;
56. struct s3 sVal3;
57. struct s4 sVal4;
58. struct s5 sVal5;
59. struct s6 sVal6;
61. printf("sVal1's size:%d\n",sizeof(sVal1));
62. printf("sVal1:%p\t%d\n",&sVal1);
64. printf("sVal2's size:%d\n",sizeof(sVal2));
65. printf("sVal2:%p\t%d\n",&sVal2);
67. printf("sVal3's size:%d\n",sizeof(sVal3));
68. printf("sVal3:%p\t%d\n",&sVal3);
70. printf("sVal4's size:%d\n",sizeof(sVal4));
71. printf("sVal4:%p\t%d\n",&sVal4);
73. printf("sVal5's size:%d\n",sizeof(sVal5));
74. printf("sVal5:%p\t%d\n",&sVal5);
76. printf("sVal5's size:%d\n",sizeof(sVal5));
77. printf("sVal5:%p\t%d\n",&sVal5);
78. }

地址供参考，主要看size，分别为：

0，8，8，8，0，0

**s1只有类型，没有变量，没有变量自然就没有空间占用（s5同)。**

**类型就是类型，和是不是结构体、联合体无关的，你的“int i;”中i不就是个变量吗？如果换成int;结果相同（这就是s6）。**

**s4和s5的做法能帮助排除干扰，将子结构体与联合体声明在外，内部直接引用，4是定义了变量，5什么都没做。**

**另外，这种做法编译的时候GCC会给你在相应的行做出提示“union\_with\_name.c:49: 警告：没有声明任何东西”**

========================================================================================================================================================================================================================================================================

以上仅属于个人心得和推测，重点在于学习思维和推理验证过程，不保证正确性与权威性。有兴趣讨论或者有发现错误的，欢迎留言交流指正。  
  
觉得好的，欢迎转载并注明出处。  
  
<http://blog.csdn.net/huqinwei987/article/details/23597091>  
  
本人博客会根据个人经验升级情况随时补充修改。