<https://www.cnblogs.com/webber1992/p/5950199.html>

转载地址：http://blog.csdn.net/firefly\_2002/article/details/7954458

一、Struct 和 Union有下列区别：

1.在存储多个成员信息时，编译器会自动给struct第个成员分配存储空间，struct 可以存储多个成员信息，而Union每个成员会用同一个存储空间，只能存储最后一个成员的信息。

2.都是由多个不同的数据类型成员组成，但在任何同一时刻，Union只存放了一个被先选中的成员，而结构体的所有成员都存在。

3.对于Union的不同成员赋值，将会对其他成员重写，原来成员的值就不存在了，而对于struct 的不同成员赋值 是互不影响的。

注：在很多地方需要对结构体的成员变量进行修改。只是部分成员变量，那么就不能用联合体Union，因为Union的所有成员变量占一个内存。eg：在链表中对个别数值域进行赋值就必须用struct.

二、实例说明

struct 简单来说就是一些相互关联的元素的集合，说是集合，其实它们在内存中的存放是有先后顺序的，并且每个元素都有自己的内存空间。那么按照什么顺序存放的呢？其实就是按你声明的变量顺序来存放的，下面先看一个例子：

**struct sTest**

**{**

**int a;  //sizeof(int) = 4**

**char b;  //sizeof(char) = 1**

**shot c；** **//sizeof(shot) = 2**

**}x;**

所以在内存中至少占用 4+1+2 = 7 byte。然而实际中占用的内存并不是7 byte，这就涉及到了字节对齐方式。

union 的不同之处就在于，它所有的元素共享同一内存单元，且分配给union的内存size 由类型最大的元素 size 来确定，如下的内存就为一个double 类型 size ：

**union uTest**

**{**

**int a;   //sizeof(int) = 4**

**double b;  //sizeof(double) = 8**

**char c;  //sizeof(char) = 1**

**}x;**

所以分配的内存 size 就是8 byte。

既然是内存共享，理所当然地，它不能同时存放多个成员的值，而只能存放其中的一个值，就是最后赋予它的值，如：

x.a = 3; x.b = 4.5; x.c = ‘A’;

这样你只看到x.c = ‘A’，而其它已经被覆盖掉，失去了意义。

eg:  Sample联合只包含其中某一个成员，要么是index，要么是price。

union Sample {

   int index;

   double price; };

若   Sample ss; ss.index =10;// 从今往后只能使用ss.index

若   Sample ss; ss.price=14.25;// 从今往后只能使用ss.price

 在union的使用中，如果给其中某个成员赋值，然后使用另一个成员，是未定义行为，后果自负。

struct成员是互相独立的，一个struct包含所有成员。  
  
C/C++ code  
struct Example  
{  
int index;  
double price;  
};  
  
Example结构包含两个成员，修改index不会对price产生影响，反之亦然。  
union的成员共享内存空间，一个union只包含其中某一个成员。

说到这里，大家应该已经明白两者最关键的区别了吧，无非就在于内存单元的分配和使用。然而要灵活地使用struct和union 还是存在许多小技巧的，比如：元素的相关性不强时，完全是可以使用union，从而节省内存size； struct和union还可以相互嵌套。

三、内存对齐方式

union u  
{  
　double a;  
　int b;  
};

union u2  
{  
　char a[13];  
　int b;  
};

union u3  
{  
　char a[13];  
　char b;  
};

cout<<sizeof(u)<<endl; // 8  
cout<<sizeof(u2)<<endl; // 16  
cout<<sizeof(u3)<<endl; // 13

 　都知道union的大小取决于它所有的成员中，占用空间最大的一个成员的大小。所以对于u来说，大小就是最大的double类型成员a了，所以sizeof(u)=sizeof(double)=8。但是对于u2和u3，最大的空间都是char[13]类型的数组，为什么u3的大小是13，而u2是16呢？关键在于u2中的成员int b。由于int类型成员的存在，使u2的对齐方式变成4，也就是说，u2的大小必须在4的对界上，所以占用的空间变成了16（最接近13的对界）。  
  
　　结论：复合数据类型，如union，struct，class的对齐方式为成员中对齐方式最大的成员的对齐方式。

顺便提一下CPU对界问题，32的C++采用8位对界来提高运行速度，所以编译器会尽量把数据放在它的对界上以提高内存命中率。对界是可以更改的，使用#pragma pack(x)宏可以改变编译器的对界方式，默认是8。C++固有类型的对界取编译器对界方式与自身大小中较小的一个。例如，指定编译器按2对界，int类型的大小是4，则int的对界为2和4中较小的2。在默认的对界方式下，因为几乎所有的数据类型都不大于默认的对界方式8（除了long double），所以所有的固有类型的对界方式可以认为就是类型自身的大小。更改一下上面的程序：

#pragma pack(2)  
union u2  
{  
　char a[13];  
　int b;  
};

union u3  
{  
　char a[13];  
　char b;  
};  
#pragma pack(8)

cout<<sizeof(u2)<<endl; // 14 由于手动更改对界方式为2，所以int的对界也变成了2，u2的对界取成员中最大的对界，也是2了，所以此时sizeof(u2)=14。  
cout<<sizeof(u3)<<endl; // 13 ，char的对界为1

**结论：**C++固有类型的对界取编译器对界方式与自身大小中较小的一个。

struct的sizeof问题  
  
　　因为对齐问题使结构体的sizeof变得比较复杂，看下面的例子：(默认对齐方式下)  
struct s1  
{  
　char a;  
　double b;  
　int c;  
　char d;  
};

struct s2  
{  
　char a;  
　char b;  
　int c;  
　double d;  
};

cout<<sizeof(s1)<<endl; // 24  
cout<<sizeof(s2)<<endl; // 16

 同样是两个char类型，一个int类型，一个double类型，但是因为对界问题，导致他们的大小不同。计算结构体大小可以采用元素摆放法，我举例子说明一下：首先，CPU判断结构体的对界，根据上一节的结论，s1和s2的对界都取最大的元素类型，也就是double类型的对界8。然后开始摆放每个元素。  
  
　　对于s1，首先把a放到8的对界，假定是0，此时下一个空闲的地址是1，但是下一个元素d是double类型，要放到8的对界上，离1最接近的地址是8了，所以d被放在了8，此时下一个空闲地址变成了16，下一个元素c的对界是4，16可以满足，所以c放在了16，此时下一个空闲地址变成了20，下一个元素d需要对界1，也正好落在对界上，所以d放在了20，结构体在地址21处结束。由于s1的大小需要是8的倍数，所以21-23的空间被保留，s1的大小变成了24。  
  
　　对于s2，首先把a放到8的对界，假定是0，此时下一个空闲地址是1，下一个元素的对界也是1，所以b摆放在1，下一个空闲地址变成了2；下一个元素c的对界是4，所以取离2最近的地址4摆放c，下一个空闲地址变成了8，下一个元素d的对界是8，所以d摆放在8，所有元素摆放完毕，结构体在15处结束，占用总空间为16，正好是8的倍数。

特例：

＃include<stdio.h>

union{

int i;

char x[2]; }a;

void main()

{

a.x[0]=10;

a.x[1]=1;

printf("%d",a.i);

 }

在联合体a中定义了两种数据类型，字符数组x以及整形变量i.其中整形变量是16位的，数组大小为2的字符数组为8X2＝16位。如此一来，编译器便会为联合体a在内存中开辟一个16位的空间，这个空间里存储联合体的数据，但是这个空间只有16位，它既是整形变量的数据，也是字符数组的数据。如果你的程序从字符数组的角度解析这个空间，那么它就是两个字符，如果你的程序从整型的角度解析这个空间，那么它就是一个整数。    
  以你的程序为例子，现在已经开辟了一个16位的空间，然后我们假定现在空间还没有被赋值，为：    
  00000000 00000000    
  那么在运行完代码    
  a.x[0] = 10;    
  a.x[1] = 1;    
  之后，16位的空间变为：    
  00000110 00000001    
  然后程序运行    
  printf("%d",a.i);    
  就是把联合体a当成一个整数来解析，而不是字符串数组。那么这样一来，程序就把这16位变成了一个完整的整数：    
  (00000001 00000110)二进制 ＝ （266）十进制