# 对齐的分类和准则

     主要基于Intel X86架构介绍结构体对齐和栈内存对齐，位域本质上为结构体类型。

     对于Intel X86平台，每次分配内存应该是从4的整数倍地址开始分配，无论是对结构体变量还是简单类型的变量。

## 3.1 结构体对齐

     在C语言中，结构体是种复合数据类型，其构成元素既可以是基本数据类型(如int、long、float等)的变量，也可以是一些复合数据类型(如数组、结构体、联合等)的数据单元。编译器为结构体的每个成员按照其自然边界(alignment)分配空间。各成员按照它们被声明的顺序在内存中顺序存储，第一个成员的地址和整个结构的地址相同。

### 对齐准则

     先来看四个重要的基本概念：

     1) 数据类型自身的对齐值：char型数据自身对齐值为1字节，short型数据为2字节，int/float型为4字节，double型为8字节。

     2) 结构体或类的自身对齐值：其成员中自身对齐值最大的那个值。

     3) 指定对齐值：#pragma pack (value)时的指定对齐值value。

     4) 数据成员、结构体和类的有效对齐值：自身对齐值和指定对齐值中较小者，即有效对齐值=min{自身对齐值，当前指定的pack值}。

     基于上面这些值，就可以方便地讨论具体数据结构的成员和其自身的对齐方式。

上面的概念非常便于理解，不过个人还是更喜欢下面的对齐准则。

     结构体字节对齐的细节和具体编译器实现相关，但一般而言满足三个准则：

     1) 结构体变量的首地址能够被其最宽基本类型成员的大小所整除；

     2) 结构体每个成员相对结构体首地址的偏移量(offset)都是成员大小的整数倍，如有需要编译器会在成员之间加上填充字节(internal adding)；

     3) 结构体的总大小为结构体最宽基本类型成员大小的整数倍，如有需要编译器会在最末一个成员之后加上填充字节{trailing padding}。

### Intel/微软C语言面试题（在最后面）

## 3.2 栈内存对齐

     在VC/C++中，栈的对齐方式不受结构体成员对齐选项的影响。总是保持对齐且对齐在4字节边界上。

【例4】

[复制代码](javascript:void(0);)

1 #pragma pack(push, 1) //后面可改为1, 2, 4, 8

2 struct StrtE{

3 char m1;

4 long m2;

5 };

6 #pragma pack(pop)

7

8 int main(void){

9 char a;

10 short b;

11 int c;

12 double d[2];

13 struct StrtE s;

14

15 printf("a address: %p\n", &a);

16 printf("b address: %p\n", &b);

17 printf("c address: %p\n", &c);

18 printf("d[0] address: %p\n", &(d[0]));

19 printf("d[1] address: %p\n", &(d[1]));

20 printf("s address: %p\n", &s);

21 printf("s.m2 address: %p\n", &(s.m2));

22 return 0;

23 }

[复制代码](javascript:void(0);)

     结果如下：

[复制代码](javascript:void(0);)

1 a address: 0xbfc4cfff

2 b address: 0xbfc4cffc

3 c address: 0xbfc4cff8

4 d**[**0**]** address: 0xbfc4cfe8

5 d**[**1**]** address: 0xbfc4cff0

6 s address: 0xbfc4cfe3

7 s.m2 address: 0xbfc4cfe4

[复制代码](javascript:void(0);)

     可以看出都是对齐到4字节。并且前面的char和short并没有被凑在一起(成4字节)，这和结构体内的处理是不同的。

     至于为什么输出的地址值是变小的，这是因为该平台下的栈是倒着“生长”的。栈是从高地址往低地址存数据。

在结构体中，综合考虑变量本身和指定的对齐值；

在栈上，不考虑变量本身的大小，统一对齐到4字节。

声明结构体类型是不占用内存空间的；类是一种用户自己定义的数据类型，因此类在定义时是不占用内存空间的。

实际上，C++对结构体进行了扩充，使之具有与类相同的特性。它们之间唯一的区别是，结构体的默认访问类型是public，而类为private。C++程序员通常使用类来实现类描述，而把结构限制为只表示纯粹的数据对象（常被称为普通老式数据（POD，Plain Old Data）结构）。

对于一般的类(非静态)来说，在定义类但还未创建对象的时候，类的所有成员(包括变量和函数)都占用着内存空间(准确地说占用着指令代码区)，但不占用堆栈空间  
  
而创建对象的时候，会根据对象的类型占用堆栈的空间(用传统模式创建对象会占用栈空间，用引用+new模式创建对象会占用堆空间，同时引用会保存在栈里)  
  
对于静态(static)类来说，静态类是不能实例化创建对象的，所有的成员都是[静态成员](https://www.baidu.com/s?wd=%E9%9D%99%E6%80%81%E6%88%90%E5%91%98&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)，也需要占用内存空间，但不在堆栈里，而是在内存的静态/全局区(这个区域用于存放所有的全局成员和[静态成员](https://www.baidu.com/s?wd=%E9%9D%99%E6%80%81%E6%88%90%E5%91%98&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao))

### Intel/微软C语言面试题

     请看下面的问题：

[复制代码](javascript:void(0);)

1 #pragma pack(8)

2 struct s1{

3 short a;

4 long b;

5 };

6 struct s2{

7 char c;

8 s1 d;

9 long long e; //VC6.0下可能要用\_\_int64代替双long

10 };

11 #pragma pack()

[复制代码](javascript:void(0);)

     问：1. sizeof(s2) = ？ 2. s2的s1中的a后面空了几个字节接着是b？

    【分析】

     成员对齐有一个重要的条件，即每个成员分别按自己的方式对齐。

     也就是说上面虽然指定了按8字节对齐，但并不是所有的成员都是以8字节对齐。其对齐的规则是：每个成员按其类型的对齐参数(通常是这个类型的大小)和指定对齐参数(这里是8字节)中较小的一个对齐，并且结构的长度必须为所用过的所有对齐参数的整数倍，不够就补空字节。

     s1中成员a是1字节，默认按1字节对齐，而指定对齐参数为8（准则4），两值中取1，即a按1字节对齐；成员b是4个字节，默认按4字节对齐，这时就按4字节对齐，所以sizeof(s1)应该为8；

     s2中c和s1中a一样，按1字节对齐。而d 是个8字节结构体，其默认对齐方式就是所有成员使用的对齐参数中最大的一个（准则2），s1的就是4。所以，成员d按4字节对齐。成员e是8个字节，默认按8字节对齐，和指定的一样，所以它对到8字节的边界上。这时，已经使用了12个字节，所以又添加4个字节的空，从第16个字节开始放置成员e。此时长度为24，并可被8(成员e按8字节对齐)整除。这样，一共使用了24个字节。

     各个变量在内存中的布局为：

     c\*\*\*aa\*\*

     bbbb\*\*\*\*

     dddddddd     ——这种“矩阵写法”很方便看出结构体实际大小！

     因此，sizeof(S2)结果为24，a后面空了2个字节接着是b。

     这里有三点很重要：

     1) 每个成员分别按自己的方式对齐，并能最小化长度；

     2) 复杂类型(如结构)的默认对齐方式是其最长的成员的对齐方式，这样在成员是复杂类型时可以最小化长度；

     3) 对齐后的长度必须是成员中最大对齐参数的整数倍，这样在处理数组时可保证每一项都边界对齐。

     还要注意，“空结构体”(不含数据成员)的大小为1，而不是0。试想如果不占空间的话，一个空结构体变量如何取地址、两个不同的空结构体变量又如何得以区分呢？