C++内存对齐总结

成员变量在类中的内存存储并不一定是连续的。它是按照编译器的设置，按照内存块来存储的，这个内存块大小的取值，就是内存对齐。

引入问题

```
#include<iostream>
using namespace std;
class test {
private :

    char c='1';//1byte
    int i;//4byte
    short s=2;//2byte
};

int main(){
    cout << sizeof(test) << endl;
    return 0;
}
```

输出：12

```
class test2 {
private:
    int i;//4byte
    char c = '1';//1byte
    short s = 2;//2byte
};

int main(){
    cout << sizeof(test2) << endl;
    return 0;
}
```

输出：8

我们可以看到。类test和test2的成员变量完全一样，只是定义顺序不一样，却造成了2个类占用内存大小不一样。而这就是编译器内存对齐的缘故

规则

1、第一个数据成员放在offset为0的地方，以后每个数据成员的对齐按照#pragma pack指定的数值和这个数据成员自身长度中，比较小的那个进行。

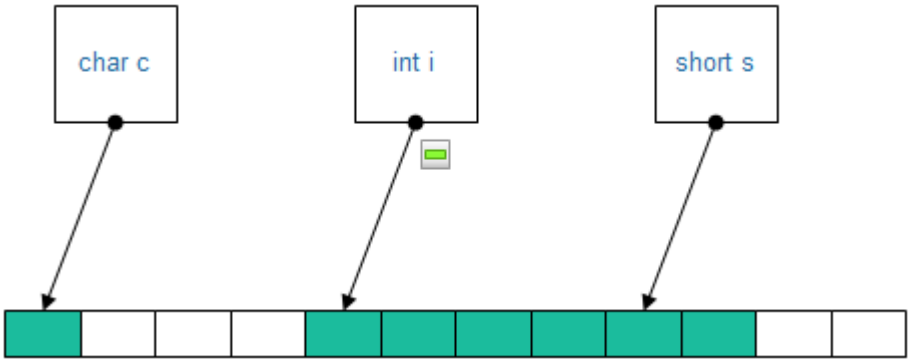
2、在数据成员完成各自对齐之后，类(结构或联合)本身也要进行对齐，对齐将按照#pragma pack指定的数值和结构(或联合)最大数据成员长度中，比较小的那个进行。

很明显#pragma pack(n)作为一个预编译指令用来设置多少个字节对齐的。值得注意的是，n的缺省数值是按照编译器自身设置，一般为8，合法的数值分别是1、2、4、8、16。

即编译器只会按照1、2、4、8、16的方式分割内存。若n为其他值，是无效的。

问题分析

（1）对于类test的内存空间是这样的：



内存分配过程：

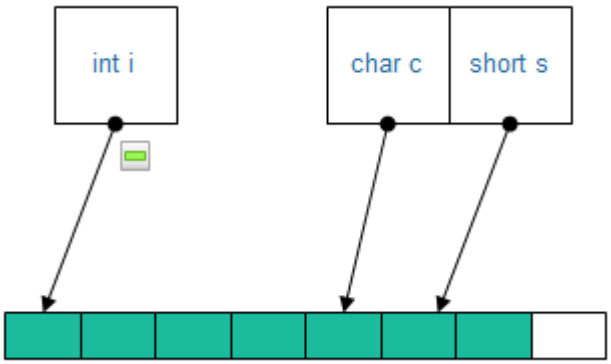
1、char和编译器默认的内存缺省分割大小比较，char比较小，分配一个字节给它。

2、int和编译器默认的内存缺省分割大小比较，int比较小，占4字节。只能空3个字节，重新分配4个字节。

3、short和编译器默认的内存缺省分割大小比较，short比较小，占2个字节，分配2个字节给它。

4、对齐结束类本身也要对齐，所以最后空余的2个字节也被test占用。

（2）对于类test2的内存空间是这样的：



1、int和编译器默认的内存缺省分割大小比较，int比较小，占4字节。分配4个字节给int。

2、char和编译器默认的内存缺省分割大小比较，char比较小，分配一个字节给它。

3、short和编译器默认的内存缺省分割大小比较，short比较小，此时前面的char分配完毕还余下3个字节，足够short的2个字节存储，所以short紧挨着。分配2个字节给short。

4、对齐结束类本身也要对齐，所以最后空余的1个字节也被test占用。

（3）使用#pragma pack(n)

```cpp
#include<iostream>
using namespace std;
#pragma pack(1)//设定为 1 字节对齐
class test {
private :

    char c='1';//1byte
    int i;//4byte
    short s=2;//2byte
};

class test2 {
private:
    int i;//4byte
    char c = '1';//1byte
    short s = 2;//2byte
};
int main(){
    cout << sizeof(test) << endl;
    cout << sizeof(test2) << endl;
    return 0;
}
```