要理解 sizeof 的要点，首先要明白字节对齐。

**一. 字节对齐**

但一般而言，满足以下两点个：  
1) 结构体每个成员相对于结构体首地址的偏移量都是本成员大小的整数倍，如不满足则加上填充字节；  
2) 结构体的总大小为结构体最宽的基本类型成员大小的整数倍，如不满足则在最后加上填充字；

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7405556)

1. struct A {
2. **int** a;
3. **char** b;
4. **short** c;
5. };
6. struct B {
7. **char** b;
8. **int** a;
9. **short** c;
10. };
11. **int** main()
12. {
13. A aa = {0xffffffff, 0x11, 0xffff};
14. B bb = {0x11, 0xffffffff, 0xffff};
15. **int** xx = sizeof(aa);
16. **int** yy = sizeof(bb);
17. return 0;
18. }

struct A {

int a;

char b;

short c;

};

struct B {

char b;

int a;

short c;

};

int main()

{

A aa = {0xffffffff, 0x11, 0xffff};

B bb = {0x11, 0xffffffff, 0xffff};

int xx = sizeof(aa);

int yy = sizeof(bb);

return 0;

}

**分析A：**

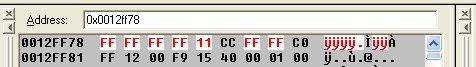
**首先第一个 int 为 4 。－－－－**

**其次 char 为 1，char 成员相对于首地址的偏移量为4， 4是char 本身大小 1 的整数倍，所以可以接着安置。－－－－－**

**然后 short 为2，short 成员相对于首地址的偏移量为5，5不是short 本身大小 2 的整数倍，所以在填充到6。－－－－－X－－**

**最后计算总大小，int 4+char 1+填充 1+ short 2 ＝ 8，8是最宽int 4 的整数倍。所以后面不再填充。**

调试结果如下：



**分析B：**

**首先第一个 char为 1 。－**

**其次 int 为 4，int 成员相对于首地址的偏移量为 1， 1显然不是 int 本身大小 4 的整数倍，所以要填充到4。－XXX**

**然后 short 为2，short 成员相对于首地址的偏移量为 8，8 是short 本身大小 2 的整数倍，所以不需要填充。－XXX－－－－－－**

**最后计算总大小，char 1 + 填充 3 + int 4 + short 2 ＝ 10，10不是最宽 int 4 的整数倍。所以后面还要再填充到12。－XXX－－－－－XX**

调试结果如下：



**二. sizeof 相关**

**1. sizeof 计算字符串长度**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7405556)

1. **char** str1[20] = "0123456789";
2. **char** str2[] = "0123456789";
3. **int** a = sizeof(str1); //20
4. **int** b = sizeof(str2); //11, 注意要还要加上 '\0'
5. **int** c = strlen(str1); //10, 注意 strlen 参数只能是char\*, 且必须以'\0'结尾

char str1[20] = "0123456789";

char str2[] = "0123456789";

int a = sizeof(str1); //20

int b = sizeof(str2); //11, 注意要还要加上 '\0'

int c = strlen(str1); //10, 注意 strlen 参数只能是char\*, 且必须以'\0'结尾

计算字符串时，sizeof 要计算 '\0'；而strlen 不计算'\0'。

**2. sizeof 计算含有虚函数的类对象大小**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7405556)

1. class A
2. {
3. public:
4. A(**int** x):a(x) {}
5. virtual void print()
6. {
7. cout << "A" << endl;
8. }
9. private:
10. **int** a;
11. };
12. class B:public A
13. {
14. public:
15. B(**int** x):A(x-1),b(x)
16. {}
17. virtual void print()
18. {
19. cout << "B" << endl;
20. }
21. private:
22. **int** b;
23. };
24. sizeof(A) == 8; // int 4 + 虚表指针 4 ＝ 8
25. sizeof(B) == 12; // int 4 + int 4 + 虚表指针 4 ＝ 12

class A

{

public:

A(int x):a(x) {}

virtual void print()

{

cout << "A" << endl;

}

private:

int a;

};

class B:public A

{

public:

B(int x):A(x-1),b(x)

{}

virtual void print()

{

cout << "B" << endl;

}

private:

int b;

};

sizeof(A) == 8; // int 4 + 虚表指针 4 ＝ 8

sizeof(B) == 12; // int 4 + int 4 + 虚表指针 4 ＝ 12

可以看出，含虚函数(不管含多少个虚函数)的类，会只有一个虚表指针。普通的成员函数不占用字节。

**3. sizeof 计算含静态成员的类大小**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7405556)

1. class A
2. {
3. public:
4. **int** a;
5. static **int** b;
6. };
7. **int** A:b = 10;
8. sizeof(A) == 4; //只计算 int a

class A

{

public:

int a;

static int b;

};

int A:b = 10;

sizeof(A) == 4; //只计算 int a

静态成员的空间不在类的实例中，而是像全局变量一样在静态存储区中，被类共享，所在不计算在内。

**4. sizeof 计算联合体大小**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7405556)

1. uniton u1
2. {
3. **double** a;
4. **int** b;
5. };
6. uniton u2
7. {
8. **char** a[13];
9. **int** b;
10. };
11. sizeof(u1) == 8; //联合体大小取所有成员中最大的一个成员的大小
12. sizeof(u2) == 16; //最大成员为13，但要按 int 的对齐方式(也就是要能整除 int)

uniton u1

{

double a;

int b;

};

uniton u2

{

char a[13];

int b;

};

sizeof(u1) == 8; //联合体大小取所有成员中最大的一个成员的大小

sizeof(u2) == 16; //最大成员为13，但要按 int 的对齐方式(也就是要能整除 int)

计算联合体时，取最大的成员，但也要考虑字节对齐。

**5. sizeof 其它方面**  
基本数据类型，char 1, shour 2, int 4, double 8, 指针 4  
函数参数，看成指针 4