

第二章 从 C 到 C++

- Bool 类型
- Const 限定符
- Const 与#define
- 结构体内存对齐
- **【1】bool 类型**

首先学习的是 **bool 类型**，它是 **C++新增的一种类型**。它表示逻辑真与假。True 表示逻辑真，false 表示逻辑假。它的存储字节数在不同的编译系统中可能有所不同，VC++中为 1 个字节。

- ❖ 声明的方式：bool result
- ❖ Bool 类型的变量可以当作整数来使用（true 一般为 1，false 为 0）。
- ❖ 把其他类型的值转换为布尔类型，非零值转换为 true,零值转换为 false.比如将 100 赋值给布尔类型的变量，就会出现截断警告，因为布尔类型占一个字节，而 int 类型是 4 个字节。

C 语言中是没有布尔类型的，我们用 int 类型进行替换。

【2】Const 限定符

Const 可以定义一个常变量，可以给一个字面常量起一个名字或者标识符，这个标识符我们就称为标识符常量，也称为常变量。标识符常量的使用方式和变量的使用方式是一样的，所以也成为常变量。

- 定义的方式

1.Const 数据类型 常变量名=常量值

2.数据类型 const 常量名=常量值

注意：**常量必须定义的时候就初始化。**

常量的值是不能被更改的。

【3】Const 和#define 的区别

Const 定义的常量和#define 定义的符号常量的区别。

- Const 所定义的常量是有类型的，而#define 定义的是没有类型的。在编译的时候，前者会进行类型安全检查，而后者只是简单的替换。
- Const 定义的变量在编译的时候分配内存，而#define 定义的常量在预编译的时候进行替换，不分配内存。
- 作用域不同，const 所定义的变量为该变量的作用域，#define 的作用域是从定义的地方到程序的结束，当然在某个点也可以用#undef 取消。
- 定义常量还可以用 enum,C++强烈建议用 const,enum 代替#define 来定义字符常量。Enum 是枚举常量。后面我们还会介绍用 inline（）内联函数，代替带参数的宏的用法。

#define 在底层的编程还是很有用的，但是对于高层次的编程，尽可能用 const enum inline() 替换 define.对于底层的编程，比如说是框架性的编程，#define 是很灵活的，

在后面介绍的 MFC 中，会用到很多的宏的用法，以及#和##的用法。宏在底层的编程是很灵活的是不可替代的。实际上泛型编程也很灵活，**泛型编程的本质也是替换**。是编译器对我们的代码进行替换。所以只要涉及到替换，就很灵活，宏也很灵活。

#define 的不好地方，它容易产生副作用：

```
//Effective C++ 3rd 的一个例子
#define CALL_WITH_MAX(a,b) f((a)>(b) ?(a):(b))
Int a=5;
Int b=0;
CALL_WITH_MAX(++a,b)      //a 累加了两次
CALL_WITH_MAX(++a,b+10)   //a 被累加一次
//在这里，调用 f 之前，a 的递增次数竟然取决于“它与谁来作比较”
```

【4】结构体内存对齐

➤ 什么是内存对齐

- (1) 编译器为每个“数据单元”安排到某个合适的位置之上。
- (2) C 和 C++ 是非常灵活的，它允许你干涉“内存对齐”。

➤ 为什么要对齐？

性能原因：在对其的地址上访问数据会比较快。

例如假设一个整数存在在一个对齐的地址之上，占 4 个字节，

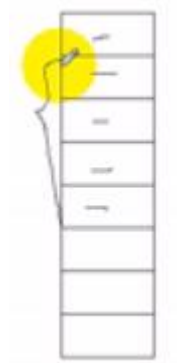


假设这个整数存放在对齐的地址，那么 CPU 在取这个整数的时候，CPU 只需要一个总线周期即可完成读取，放到寄存器当中。



前面一列是内存，后面的两列是读取后的寄存器。

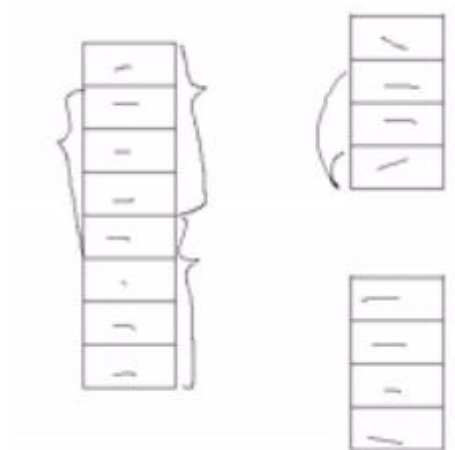
假设一个整数不是按照内存对齐存储的，那么 CPU 在读取这个整数的时候，需要经过两次总线周期，



因为一些 CPU 在数据读取数据的时候，一些是从奇地址开始读取，有的是从偶地址开始读取，



对于 32 位的系统，每个总线周期，总是能读取四个字节，两次总线周期才能读取到这个数据，

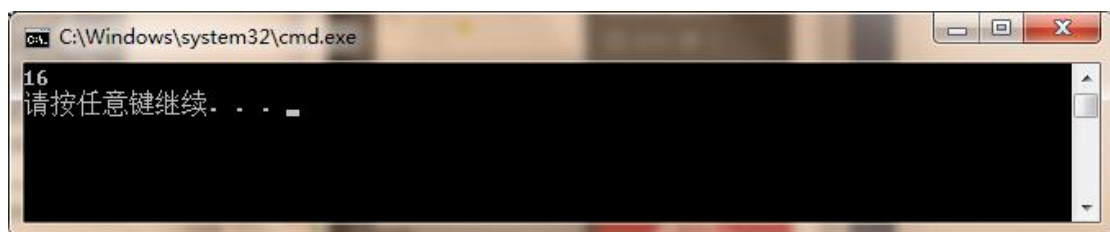
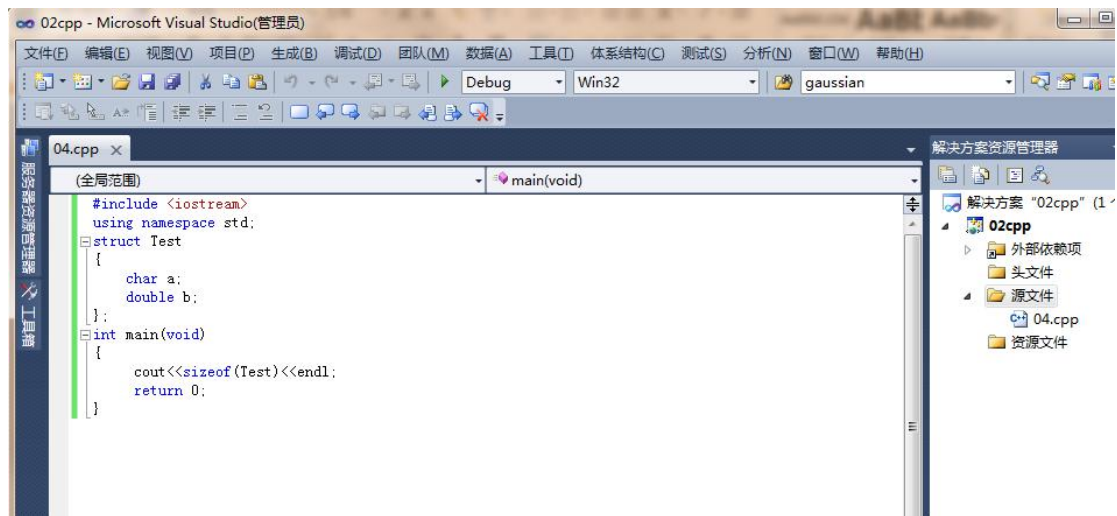


还要将这个两个寄存器合并到一个寄存器当中，第一个寄存器进行一个移位的操作，第二个寄存器向下移动数据，然后两个寄存器合并。这样的话效率就会比较的低。如果我们在对其的地址上存取数据，那么 CPU 在访问数据的时候，效率就会比较高。

接下来我们要讲解的是**如何对齐**。

对齐的一些规则，这里我们直接在代码当中体现吧。

在 C++ 当中**结构体也是类**，后面会进行讲解。

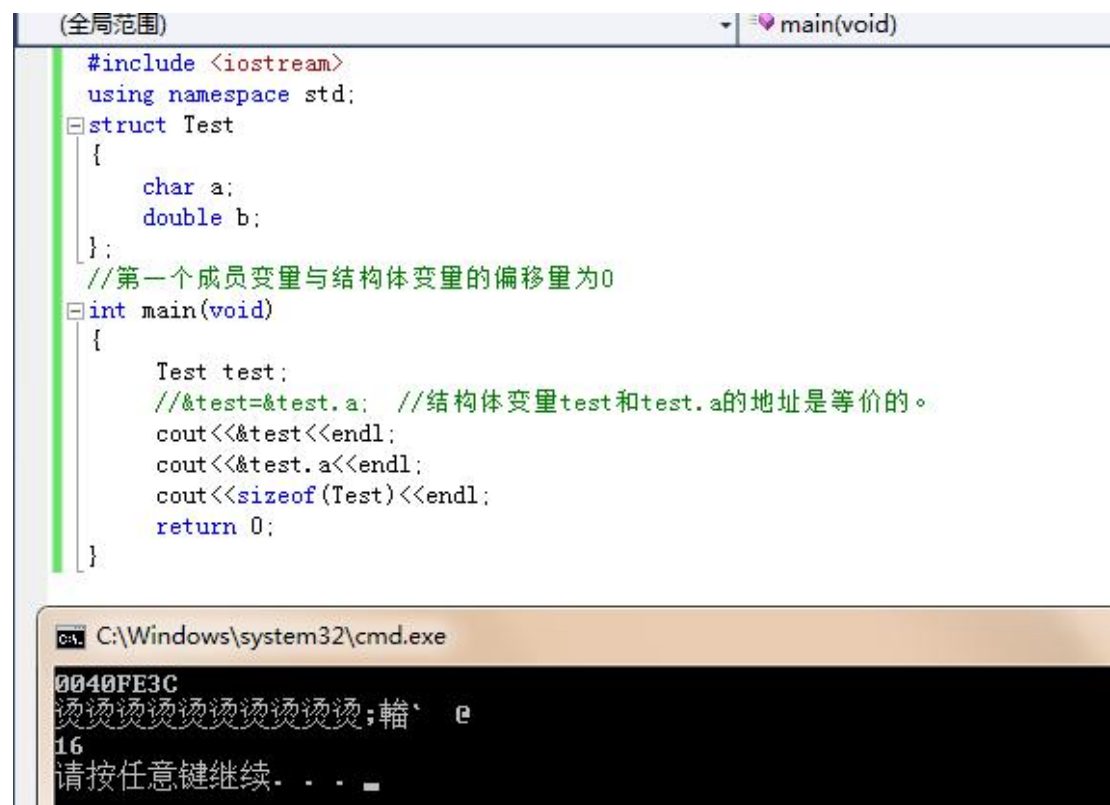


我们可以看到大小为 16，可能有些同学它的直观的感觉是应该是 9 个字节，因为 char a 是一个字节，double 是 8 个字节。现在我来解释一下结构体对齐，它的一些规则。

- 第一个规则是第一个成员与结构体变量的偏移量为零。也就是说第一个成员变量总是存放在偏移量为零的地址之上。也就是说结构体变量的地址和第一个成员变量的地址是一样的。



将两个地址输出查看一下，输出结果如下：

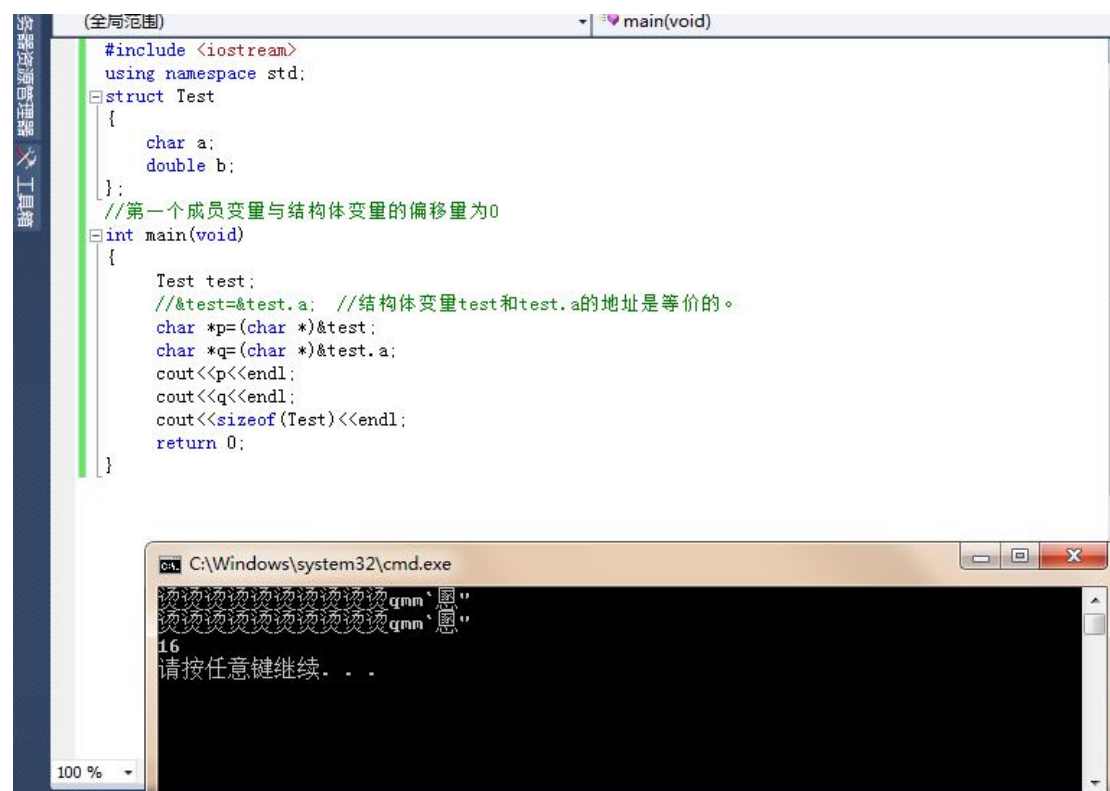


```
(全局范围) main(void)
#include <iostream>
using namespace std;
struct Test
{
    char a;
    double b;
};
//第一个成员变量与结构体变量的偏移量为0
int main(void)
{
    Test test;
    //&test=&test.a; //结构体变量test和test.a的地址是等价的。
    cout<<&test<<endl;
    cout<<&test.a<<endl;
    cout<<sizeof(Test)<<endl;
    return 0;
}
```

C:\Windows\system32\cmd.exe

0040FE3C
请按任意键继续. . .

这里按照一定格式输出的时候，除了问题，我们定义一个指针吧。



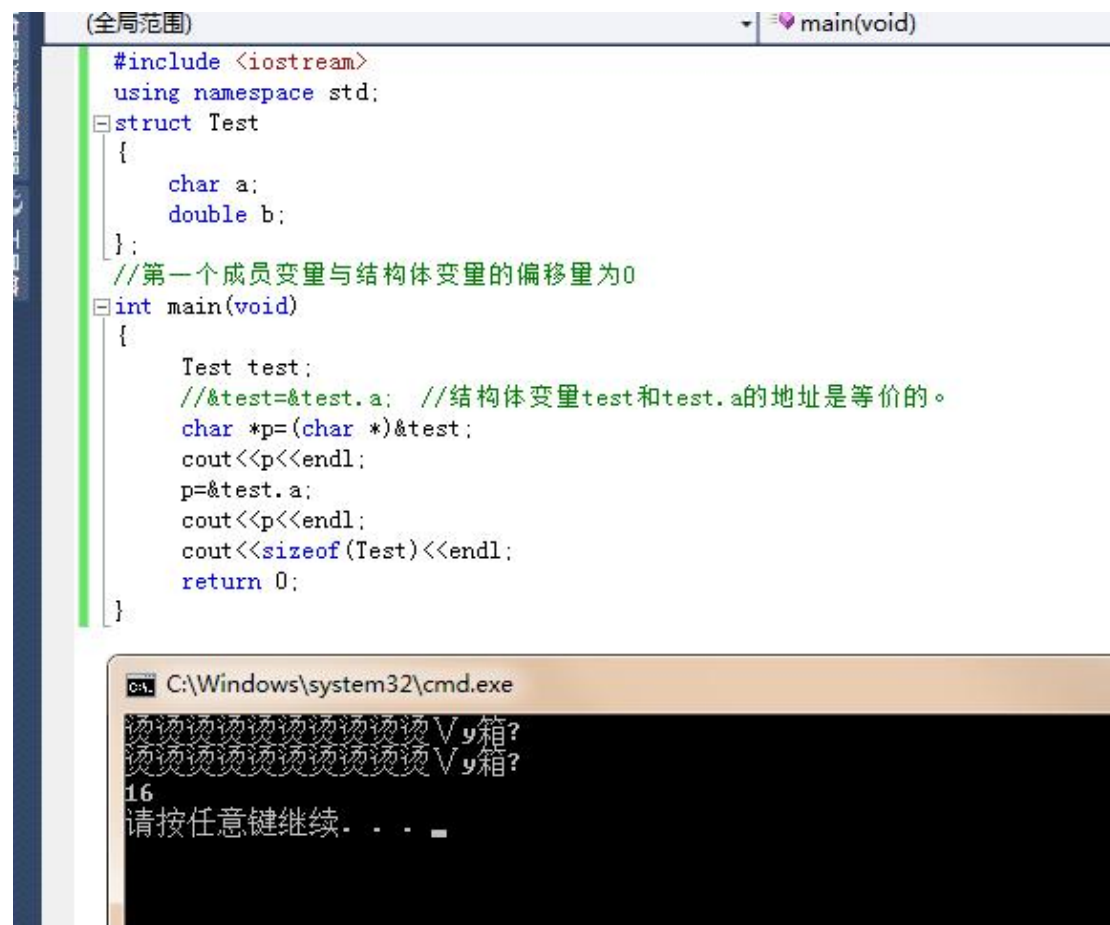
```
(全局范围) main(void)
#include <iostream>
using namespace std;
struct Test
{
    char a;
    double b;
};
//第一个成员变量与结构体变量的偏移量为0
int main(void)
{
    Test test;
    //&test=&test.a; //结构体变量test和test.a的地址是等价的。
    char *p=(char *)&test;
    char *q=(char *)&test.a;
    cout<<p<<endl;
    cout<<q<<endl;
    cout<<sizeof(Test)<<endl;
    return 0;
}
```

C:\Windows\system32\cmd.exe

0040FE3C
请按任意键继续. . .

输出是相等的。

另一种实现:



The screenshot shows a C++ IDE with a code editor and a console window. The code defines a struct 'Test' with a 'char' member 'a' and a 'double' member 'b'. The 'main' function creates a 'Test' variable 'test', prints its address using 'cout', and then prints the address of 'test.a' using 'cout'. A comment indicates that the addresses are equivalent. The console output shows two identical memory addresses, followed by the number '16' and a prompt to press any key to continue.

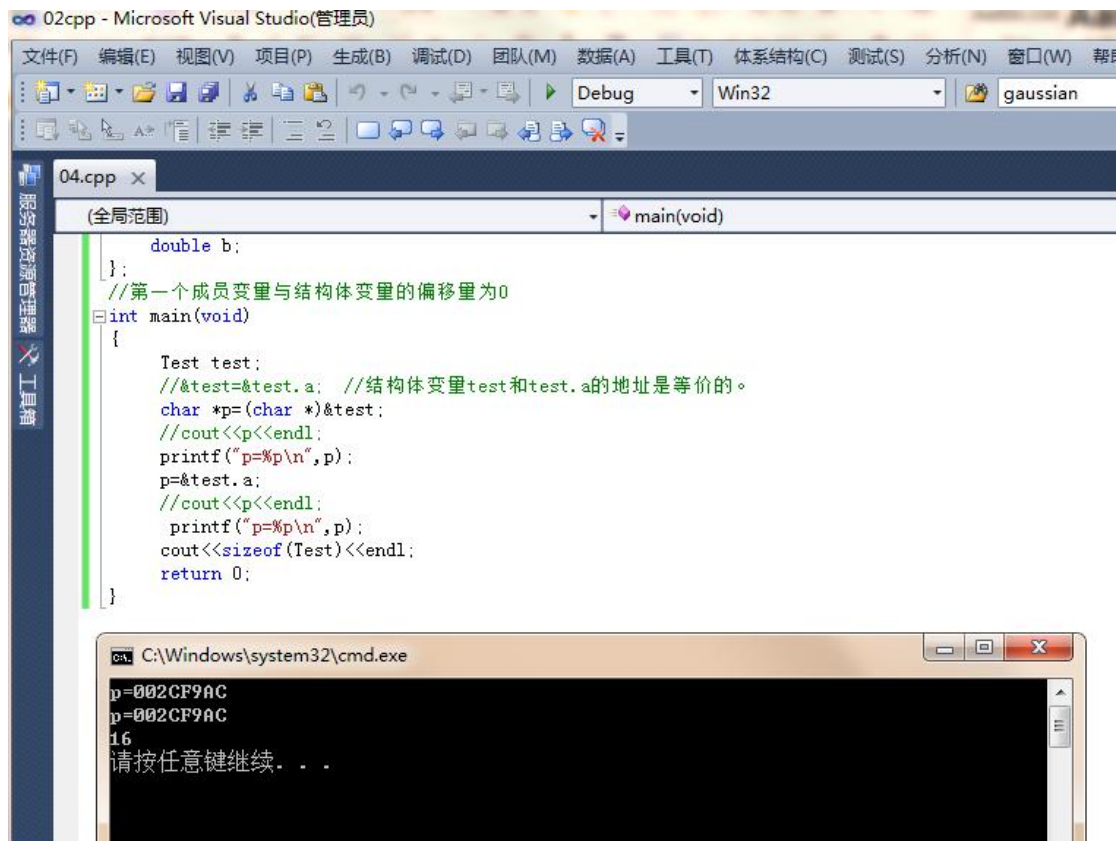
```
(全局范围) main(void)
#include <iostream>
using namespace std;
struct Test
{
    char a;
    double b;
};
//第一个成员变量与结构体变量的偏移量为0
int main(void)
{
    Test test;
    //&test=&test.a; //结构体变量test和test.a的地址是等价的。
    char *p=(char *)&test;
    cout<<p<<endl;
    p=&test.a;
    cout<<p<<endl;
    cout<<sizeof(Test)<<endl;
    return 0;
}
```

C:\Windows\system32\cmd.exe

16

请按任意键继续. . .

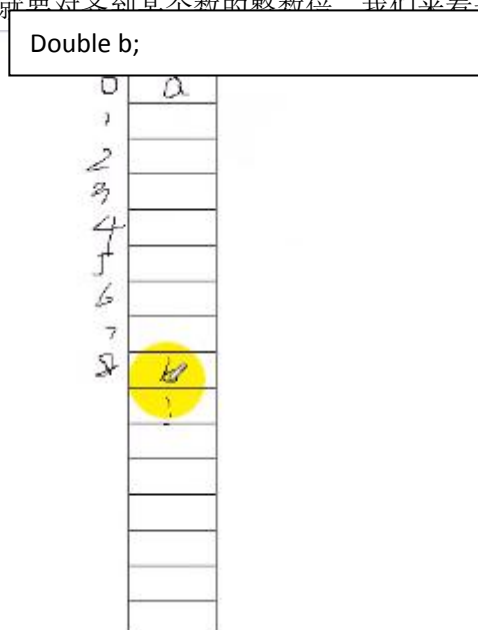
可以看到他们两个是相等的，他可能在输出的时候的问题，可能是 `cout` 输出的问题。我们可以使用 `printf()` 输出吧。因为 `printf` 的输出是按照一定方式的输出。并不是输出地址。如下：



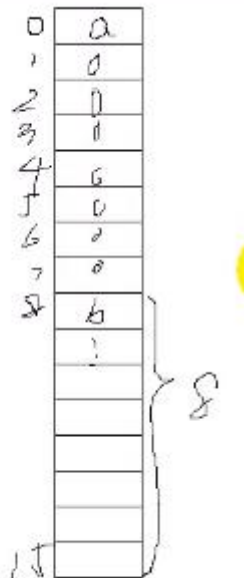
这里我们就知道地址是一样的。也就是第一个成员的地址就是结构体变量的地址。

➤ 接下来其他成员要对齐到某个数的整数倍的地址。什么意思呢？

就要对齐到某个数的整数倍。我们来看一下这个例子，它是怎么存放点。

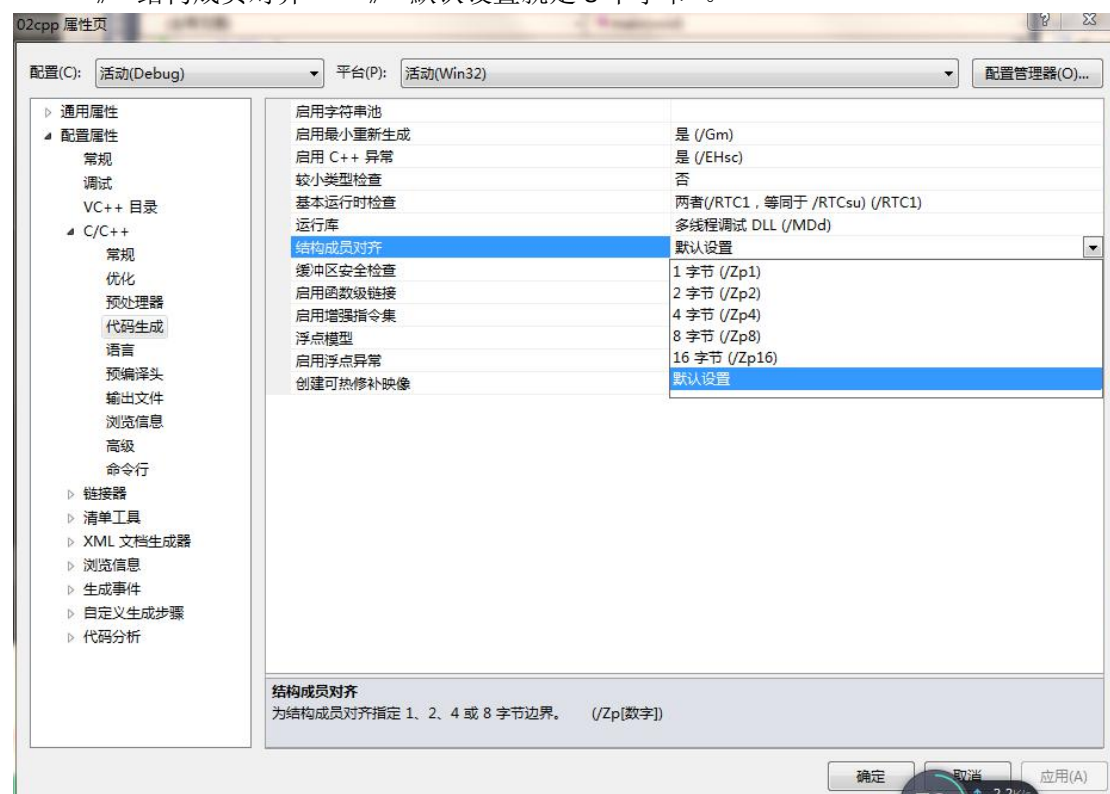


这个数就是 8,在这个例子中，也就是从 8 开始存放 b.



a 变量的值，整个 1-7 填充的是 0.那么这个结构体的大小就是 16.那么这个数（也就是前面说的某个数）是怎么计算的呢？我们把这个数叫做**对齐数**吧。接下来我们看一下对齐数是怎么计算的。

- **对齐数**取编译器预设的一个对齐整数与该成员大小的较小值。在 Visual C++ 中这个对齐整数默认值是 8,我们可以在下面看到，工程右键”属性”，-----》”C/C++”-----》”代码生成”---》”结构成员对齐” ----》”默认设置就是 8 个字节”。

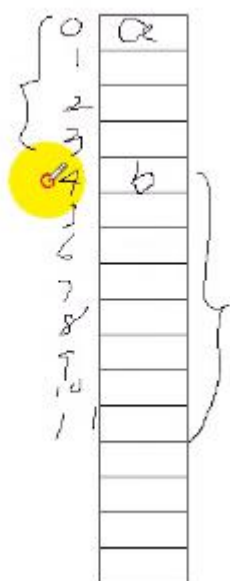


Double 类型的占 8 个字节，所以还是 8 个字节。接着我们将这个字节数修改来说明这个问题。我们把这个值修改为 4,然后重新编译运行一下。


```
04.cpp x
(全局范围) main(void)
#include <iostream>
using namespace std;
#include <stdio.h>
struct Test
{
    char a;
    double b;
};
//第一个成员变量与结构体变量的偏移量为0
int main(void)
{
    Test test;
    //&test=&test.a; //结构体变量test和test.a的地址是等价的。
    char *p=(char *)&test;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    p=&test.a;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    cout<<sizeof(Test)<<endl;
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
p=001CFC0C
p=001CFC0C
12
请按任意键继续. . .
```

修改之后的对齐数应该是多少呢？预设的对齐数是 4，成员变量的大小等于 8.所以对齐数应该是 4.那么 a 存放到了 0 的地址。b 从 4 开始存，存了 8 个字节。

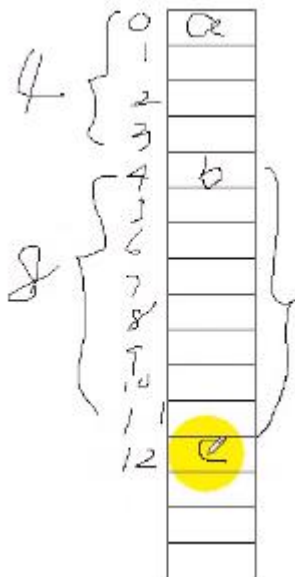


$$4+8=12$$

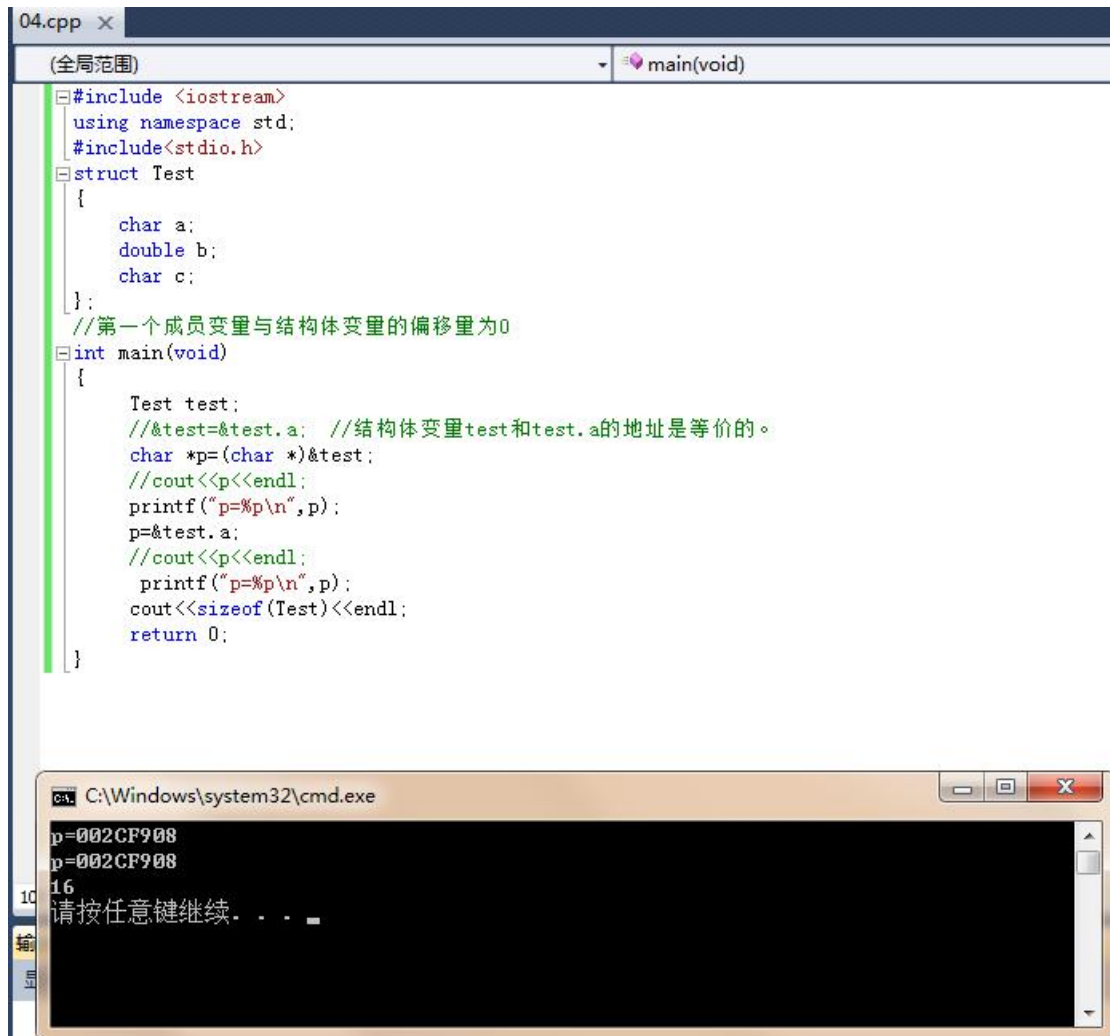
接下来我们在进行一下变化，结构体新加 char c;

```
#include <iostream>
using namespace std;
#include<stdio.h>
struct Test
{
    char a;
    double b;
    char c;
};
//第一个成员变量与结构体变量的偏移量为0
int main(void)
{
    Test test;
    //&test=&test.a; //结构体变量test和test.a的地址是等价的。
    char *p=(char *)&test;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    p=&test.a;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    cout<<sizeof(Test)<<endl;
    return 0;
}
```

因为 char 占一个字节，预设对齐数是 4，所以最终的对齐数是 1.这样结构体占 13 个字节。



我们看一下是否是 13 个字节呢？



The image shows a C++ IDE with a code editor and a command prompt window. The code editor displays the following code:

```
04.cpp x
(全局范围) main(void)
#include <iostream>
using namespace std;
#include<stdio.h>
struct Test
{
    char a;
    double b;
    char c;
};
//第一个成员变量与结构体变量的偏移量为0
int main(void)
{
    Test test;
    //&test=&test.a; //结构体变量test和test.a的地址是等价的。
    char *p=(char *)&test;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    p=&test.a;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    cout<<sizeof(Test)<<endl;
    return 0;
}
```

The command prompt window shows the output of the program:

```
C:\Windows\system32\cmd.exe
p=002CF908
p=002CF908
16
请按任意键继续. . .
```

我们发现它不是等于 13 个字节,为什么不是 13 个字节呢? 原因在于整个结构体的大小还要进行对齐,还要扩充成某个数字的整数倍,这里的数字就是 4,要变成 4 的整数倍。要扩充成 16.

➤ 结构体总大小为最大对齐数的整数倍。

将默认对齐数改为 8, 然后看这个结构体的大小为:

```
(主函数) main(void)
#include <iostream>
using namespace std;
#include <stdio.h>
struct Test
{
    char a;
    double b;
    char c;
};
//第一个成员变量与结构体变量的偏移量为0
int main(void)
{
    Test test;
    //&test=&test.a; //结构体变量test和test.a的地址是等价的。
    char *p=(char *)&test;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    p=&test.a;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    cout<<sizeof(Test)<<endl;
    return 0;
}
```

```
C:\Windows\system32\cmd.exe
p=002EFC5C
p=002EFC5C
24
请按任意键继续. . .
```

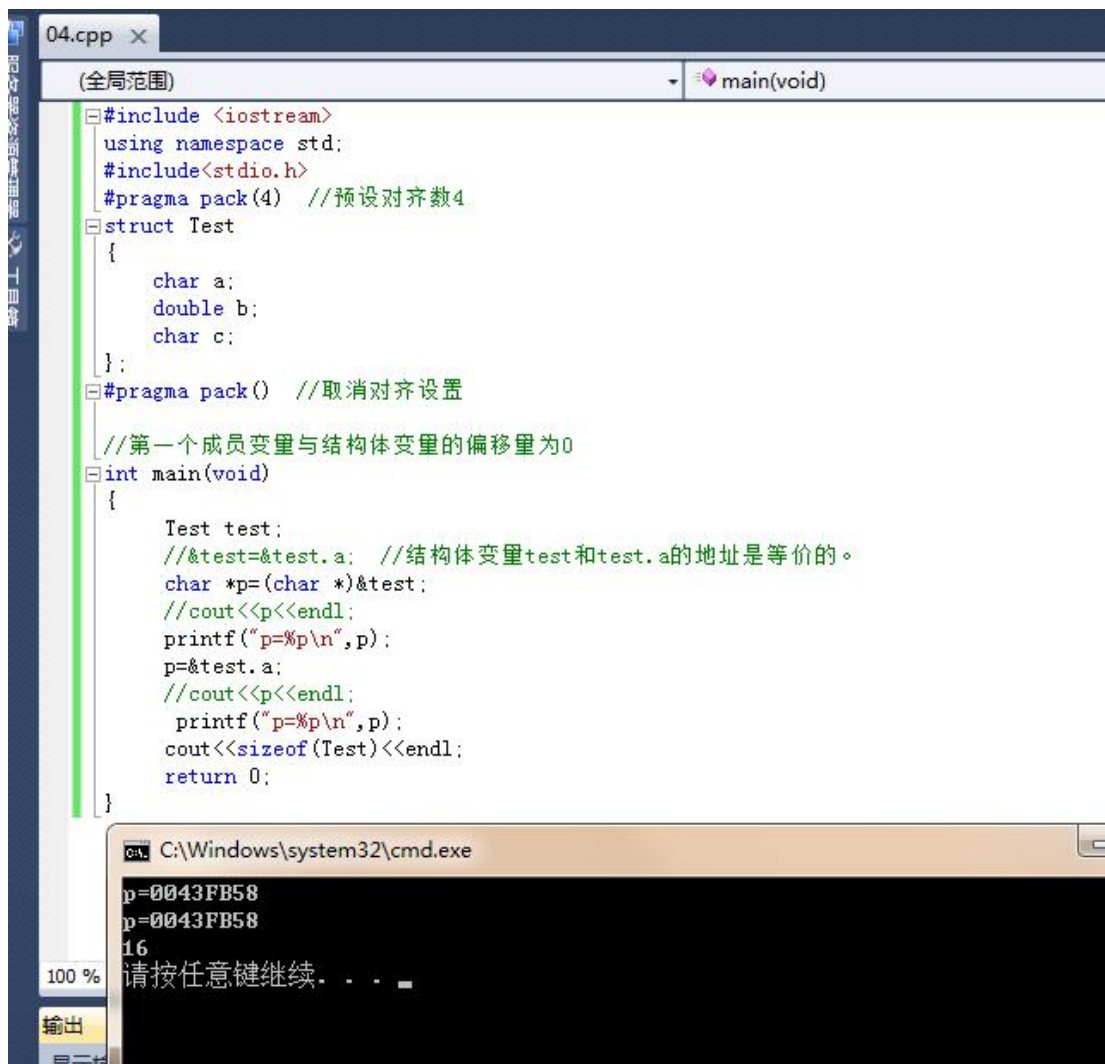
a 从零开始存 0-7

b 从 8 开始存, 8-15

c 从 16 开始,16-23

结构体的总大小还要对齐到最大对齐数的整数倍,当前的最大对齐数是 8, 17 扩充到 8 个整数倍应该是 24.

另外我们也可以通过这条指令来更改预设的对齐数:



```
04.cpp x
(全局范围) main(void)

#include <iostream>
using namespace std;
#include<stdio.h>
#pragma pack(4) //预设对齐数4

struct Test
{
    char a;
    double b;
    char c;
};

#pragma pack() //取消对齐设置

//第一个成员变量与结构体变量的偏移量为0
int main(void)
{
    Test test;
    //&test=&test.a; //结构体变量test和test.a的地址是等价的。
    char *p=(char *)&test;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    p=&test.a;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    cout<<sizeof(Test)<<endl;
    return 0;
}
```

100 %

输出

显示转

C:\Windows\system32\cmd.exe

p=0043FB58
p=0043FB58
16
请按任意键继续. . .

如果设置为#pragma pack(2)

```
#include <iostream>
using namespace std;
#include<stdio.h>
#pragma pack(2) //预设对齐数4
struct Test
{
    char a;
    double b;
    char c;
};
#pragma pack() //取消对齐设置

//第一个成员变量与结构体变量的偏移量为0
int main(void)
{
    Test test;
    //&test=&test.a; //结构体变量test和test.a的地址是等价的。
    char *p=(char *)&test;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    p=&test.a;
    //cout<<p<<endl;
    printf("p=%p\n",p);
    cout<<sizeof(Test)<<endl;
    return 0;
}
```

我们将这段代码放到 Gcc 中运行呢？也是 12 个字节，答案是一样的，另外我们也可以看一下，编译器预设的对齐数是 4.另外一点是 G++中，设置的对齐整数只能取 1，2，4，不能取 8,16,，改成 8 也没有用，比如说我改成 8，他也不会等于 24 个字节。还是 16 个字节。Vsual C++可以设置成 1，2，4，8,16.