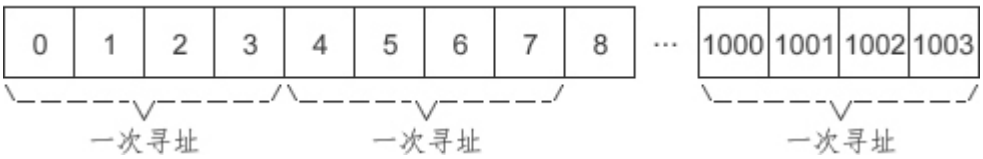
C语言内存对齐，提高寻址效率

计算机内存是以字节（Byte）为单位划分的，理论上CPU可以访问任意编号的字节，但实际情况并非如此。

CPU 通过地址总线来访问内存，一次能处理几个字节的数据，就命令地址总线读取几个字节的数据。32 位的 CPU 一次可以处理4个字节的数据，那么每次就从内存读取4个字节的数据；少了浪费主频，多了没有用。64位的处理器也是这个道理，每次读取8个字节。

以32位的CPU为例，实际寻址的步长为4个字节，也就是只对编号为 4 的倍数的内存寻址，例如 0、4、8、12、1000 等，而不会对编号为 1、3、11、1001 的内存寻址。如下图所示：



这样做可以以最快的速度寻址：不遗漏一个字节，也不重复对一个字节寻址。

对于程序来说，一个变量最好位于一个寻址步长的范围内，这样一次就可以读取到变量的值；如果跨步长存储，就需要读取两次，然后再拼接数据，效率显然降低了。

例如一个 int 类型的数据，如果地址为 8，那么很好办，对编号为 8 的内存寻址一次就可以。如果编号为 10，就比较麻烦，CPU需要先对编号为 8 的内存寻址，读取4个字节，得到该数据的前半部分，然后再对编号为 12 的内存寻址，读取4个字节，得到该数据的后半部分，再将这两部分拼接起来，才能取得数据的值。

将一个数据尽量放在一个步长之内，避免跨步长存储，这称为内存对齐。在32位编译模式下，默认以4字节对齐；在64位编译模式下，默认以8字节对齐。

为了提高存取效率，编译器会自动进行内存对齐，请看下面的代码：

```
01.  #include <stdio.h>
02.  #include <stdlib.h>
03.
04.  struct{
05.      int a;
06.      char b;
07.      int c;
08.  }t={ 10, 'C', 20 };
09.
10.  int main(){
```

```
    printf("length: %d\n", sizeof(t));
    printf("&a: %X\n&b: %X\n&c: %X\n", &t.a, &t.b, &t.c);

    system("pause");
    return 0;
}
```

在32位编译模式下的运行结果：
length: 12
&a: B69030
&b: B69034
&c: B69038

如果不考虑内存对齐，结构体变量 t 所占内存应该为 4+1+4 = 9 个字节。考虑到内存对齐，虽然成员 b 只占用1个字节，但它所在的寻址步长内还剩下 3 个字节的空间，放不下一个 int 型的变量了，所以要把成员 c 放到下一个寻址步长。剩下的这3个字节，作为内存填充浪费掉了。请看下图：

编译器之所以要内存对齐，是为了更加高效的存取成员 c，而代价就是浪费了3个字节的空间。

除了结构体，变量也会进行内存对齐，请看下面的代码：

```
#include <stdio.h>
#include <stdlib.h>

int m;
char c;
int n;

int main(){
    printf("&m: %X\n&c: %X\n&n: %X\n", &m, &c, &n);
    system("pause");
    return 0;
}
```

在VS下运行：
&m: DE3384
&c: DE338C
&n: DE3388

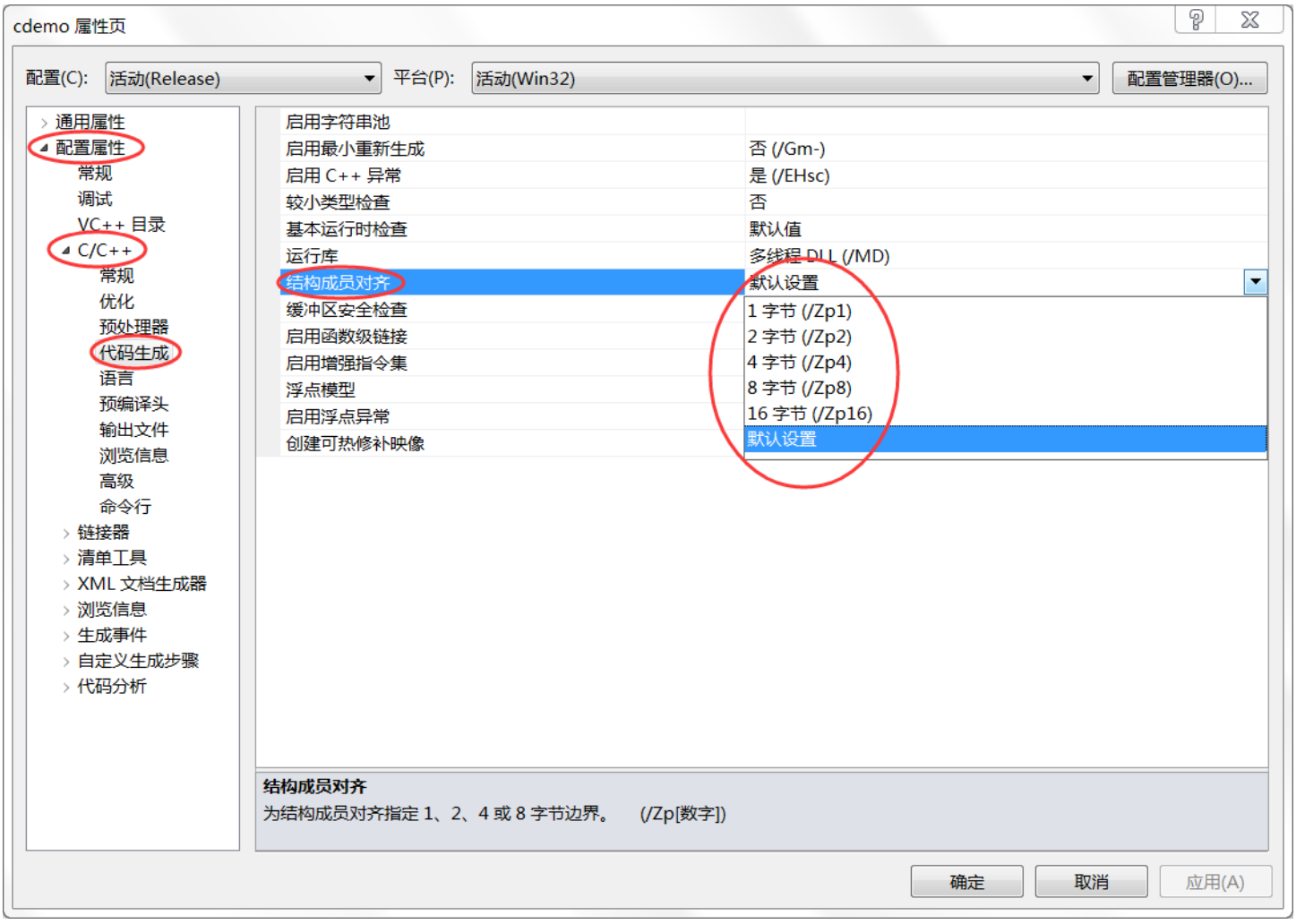
可见它们的地址都是4的整数倍，并相互挨着。

经过笔者测试，对于全局变量，GCC在 Debug 和 Release 模式下都会进行内存对齐，而VS只有在 Release 模式下才会进行对齐。而对于局部变量，GCC和VS都不会进行对齐，不管是Debug模式还是Release模式。

改变对齐方式

内存对齐虽然和硬件有关，但是决定对齐方式的是编译器，如果你的硬件是64位的，却以32位的方式编译，那么还是会按照4个字节对齐。

对齐方式可以通过编译器参数修改，以VS2010为例，更改对齐方式的步骤为：项目 --> 属性 --> C/C++ --> 代码生成 --> 结构成员对齐，如下图所示：



最后需要说明的是：内存对齐不是C语言的特性，它属于计算机的运行原理，C++、Java、Python等其他编程语言同样也会有内存对齐的问题。