C语言反汇编 _ switch _ case

---

0 . 说明

> *看滴水逆向视频总结笔记*

> *编译器VC++6.0*

调试switch反汇编时，要先思考，switch_case语句与if_else有什么差别，如果没有差别，那为什么语法里要多一个switch_case语句呢？？

我觉得滴水里的那句话说话说得很好:`要调试高级语言，就从编译器的角度出发；要调试汇编，就得从CPU的角度出发。`(就是如何提高效率)

> *所以可以思考设置switch_case语句是否比if_else语句效率高？高在什么地方？*

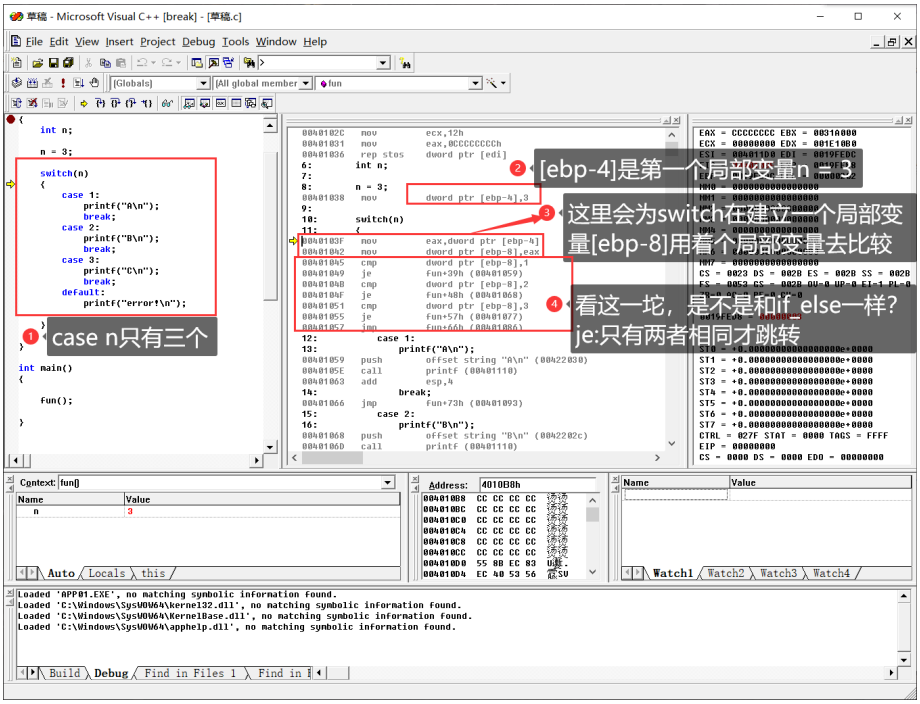
switch反汇编有三种情况，根据`case n:`的种类，编译器会自己“思考”，生成不同的switch汇编层代码。

1.情况一：汇编层和if_else相同的

这两种情况下不建议使用switch语句，直接使用if_else就可以了。

a.当`case n:`个数小于等于三时,无论是否连续。

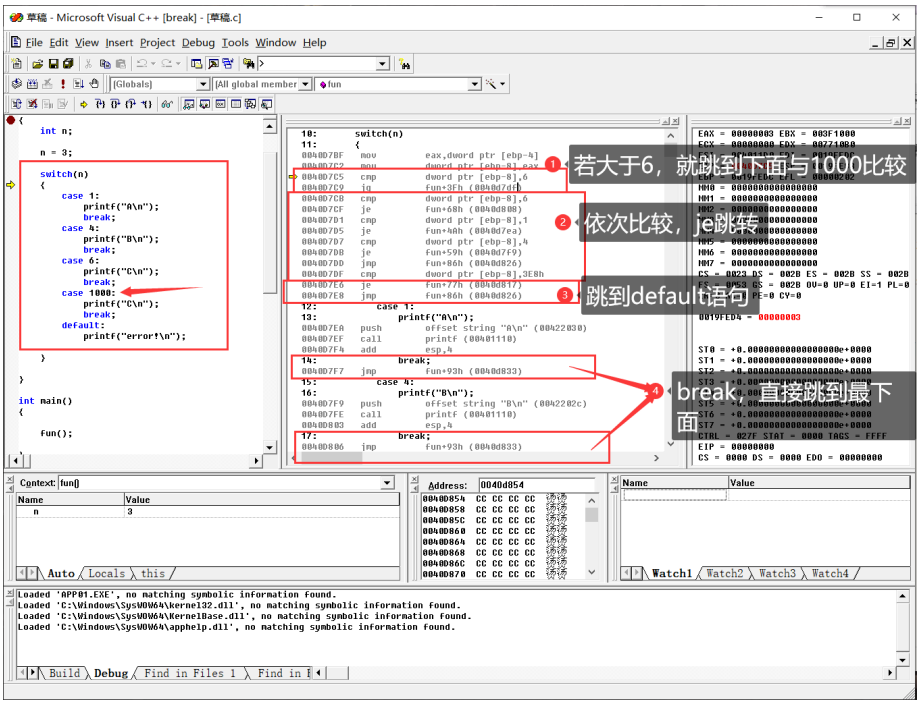
若加上`default:`则小于等于四，先别管`case n:`大于三且连续会怎样，反正不一样，会生成表格装载地址，不同情况还会生成对应`大表`和`小表`。



注意看红方框4里，je要跳转的地址，就是`case n:`对应的地址。

最后面如果都不相等，都没有跳转，直接jmp到`default:`语句

b.当`case n:`个数大于三，但n不连续，且相差较大时。



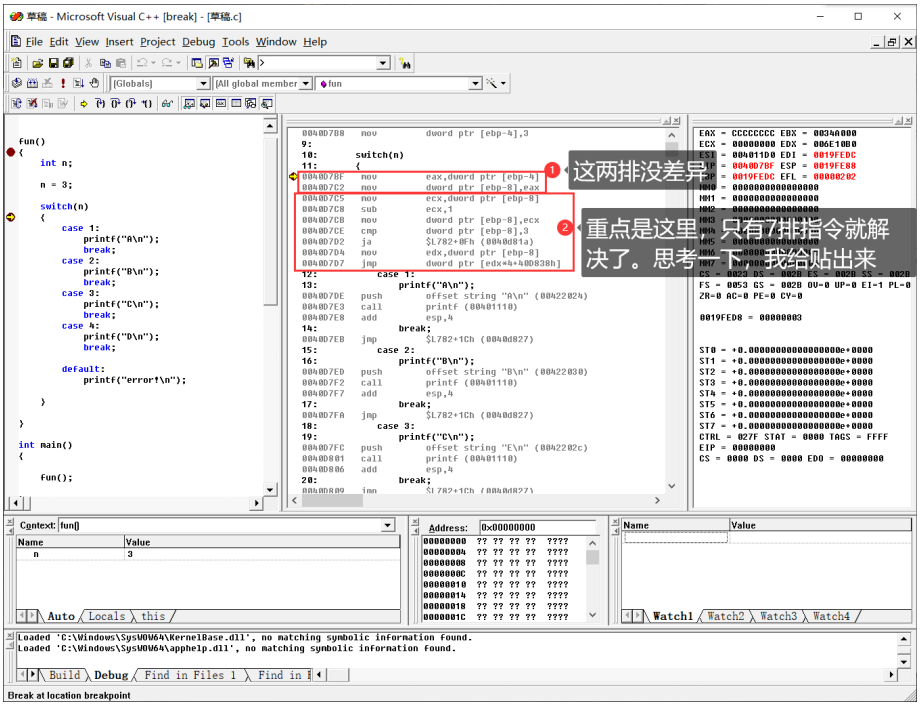
你会发现，其实还是比if_else智能一点点，因为1、4、6相差较小，1000相差特别大，switch会优先和6比较，若大于6，就直接去和1000比较，入小于或等于，再和6、1、4依次比较，而且你注意，`jg`之后先和6比较`je`，而不是按照源代码顺序1、4、6。

2.情况二：内存中生成大表装地址

> *要理解下面部分内容，前提是明白C语言反汇编中数组是如何寻址的*

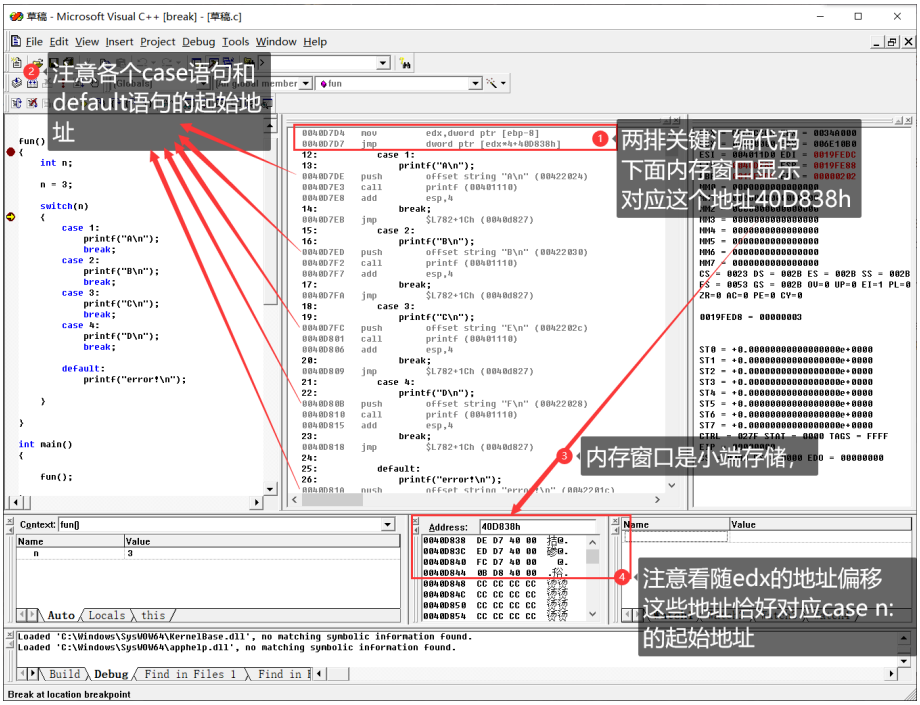
a.当case n:个数大于三，且连续时。

所谓连续，不管case n的书写顺序，只管其中所有的n合在一起，是否连续。




```
0040D7BF    mov         eax,dword ptr [ebp-4]
0040D7C2    mov         dword ptr [ebp-8],eax
//取局部变量n放入[ebp-8]，建立switch的局部变量N
（为了辨认，我用N表示，其实就是n，只不过是switch函数的参数）

0040D7C5    mov         ecx,dword ptr [ebp-8]
0040D7C8    sub         ecx,1
0040D7CB    mov         dword ptr [ebp-8],ecx
//将其减1

0040D7CE    cmp         dword ptr [ebp-8],3
0040D7D2    ja          $L782+0Fh (0040d81a)
//减1后与3比较，相当于原N与4比较
//ja大于跳转，说明你判断是否要直接进入default语句

0040D7D4    mov         edx,dword ptr [ebp-8]
0040D7D7    jmp         dword ptr [edx*4+40D838h]
//这两排是 关键汇编代码
//[edx*4+地址]这个操作是不是很想数值的寻址呢？
//这个地址下面会有啥呢？？？
```

那查看一下这个内存地址(40D838h)装的是啥？



查看内存数据

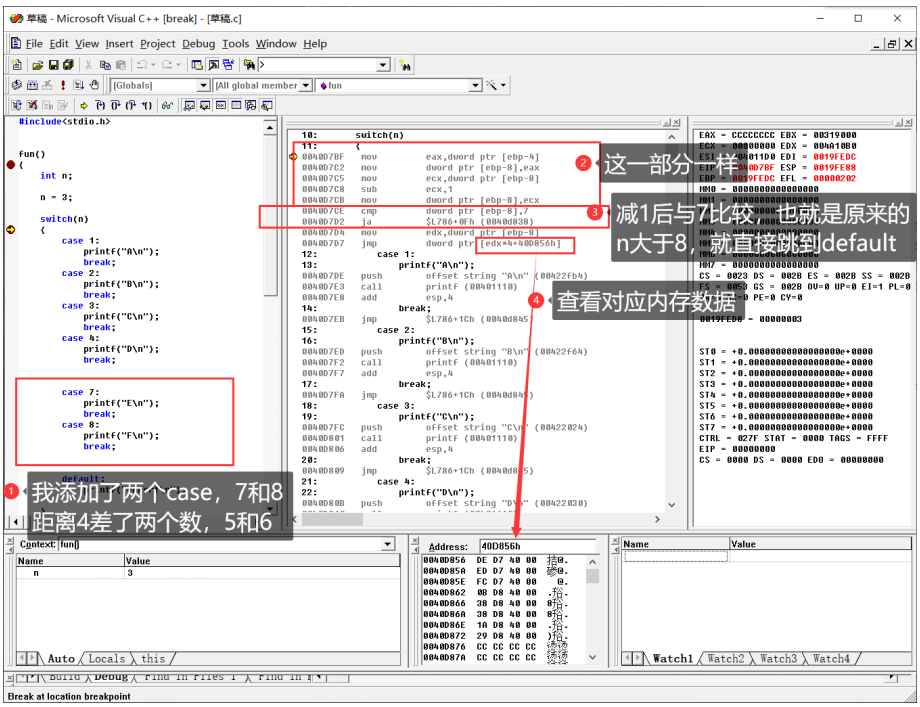
```
Address:    40D838h//内存对应地址的数据
//由于是小端存储，我在后面写上对应的正常数据
0040D838  DE D7 40 00  //00 40 D7 DE//case 1:起始地址
0040D83C  ED D7 40 00  //00 40 D7 ED//case 2:起始地址
0040D840  FC D7 40 00  //00 40 D7 FC//case 3:起始地址
0040D844  0B D8 40 00  //00 40 D8 0B//case 4:起始地址
0040D848  CC CC CC CC
0040D84C  CC CC CC CC
```

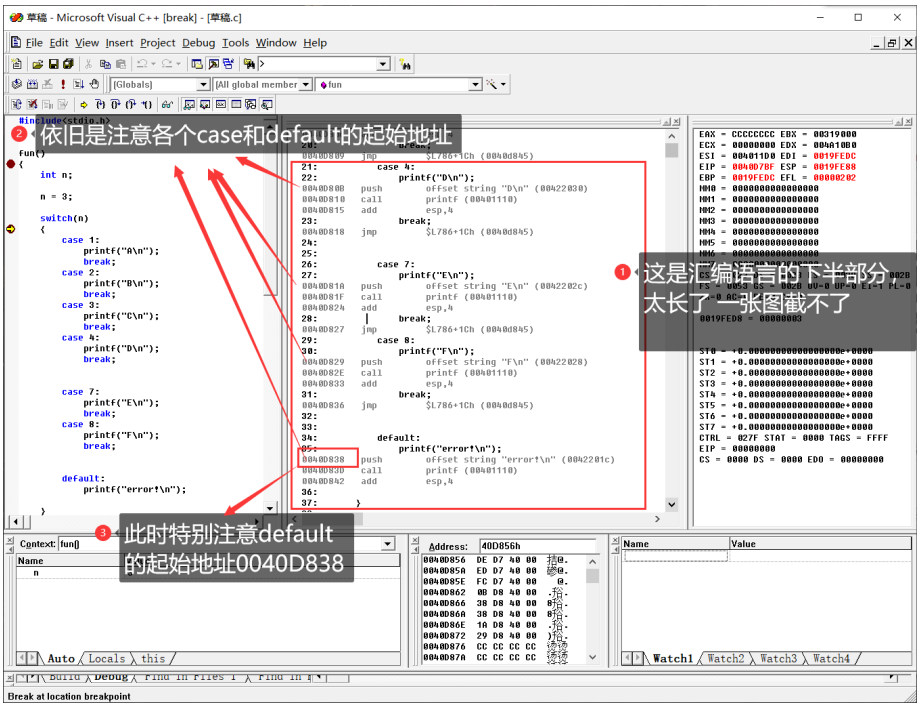
N=3时，减去1，对于2，edx对于2后，寻址结果是地址：00 40 D7 FC，还是`case 3:`的起始地址。

所以那两句关键汇编代码，形似数组寻址，其实也差不多，将原本的N减一，就是为了方便寻址，只不过对应内存中装的是地址。

内存中的这一部分数据，就称为switch生成的大表，表内装有地址。

b.当`case n:`个数大于三，n不连续，但相差较小。





查看内存数据

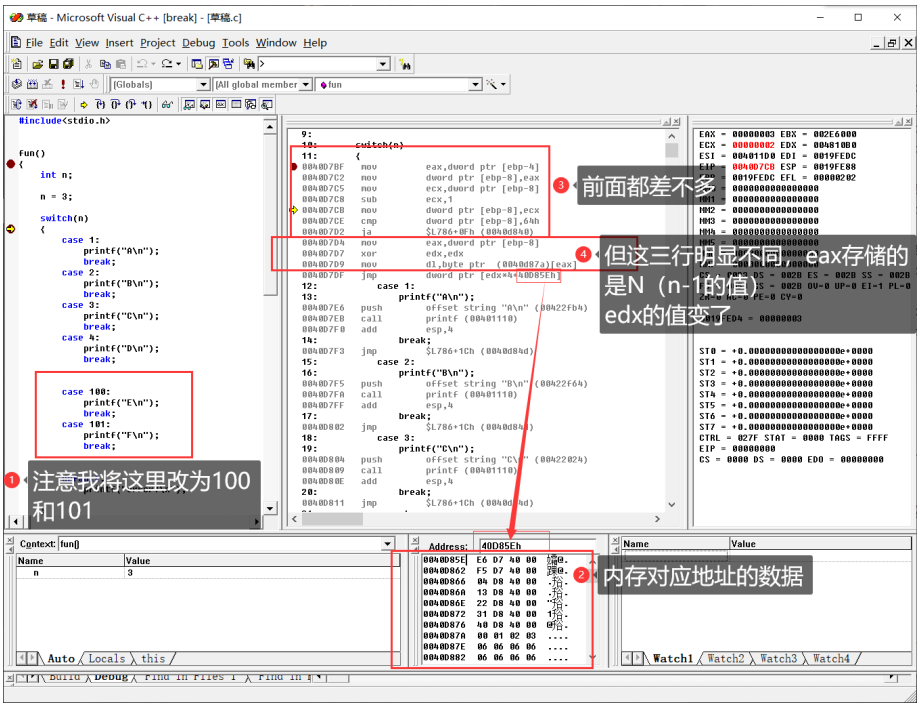
```
Address:    40D856h
0040D856  DE D7 40 00  //00 40 D7 DE//case 1:入口地址
0040D85A  ED D7 40 00  //00 40 D7 ED//case 2:入口地址
0040D85E  FC D7 40 00  //00 40 D7 FC//case 3:入口地址
0040D862  0B D8 40 00  //00 40 D8 0B//case 4:入口地址

//n=5时，edx=5-1=4，此时对于内存的偏移正好是default地址
0040D866  38 D8 40 00  //00 40 D8 38//default入口地址

//n=6时，edx=6-1=5，此时对于内存的偏移正好是default地址
0040D86A  38 D8 40 00  //00 40 D8 38//default入口地址

0040D86E  1A D8 40 00  //00 40 D8 1A//case 7:入口地址
```

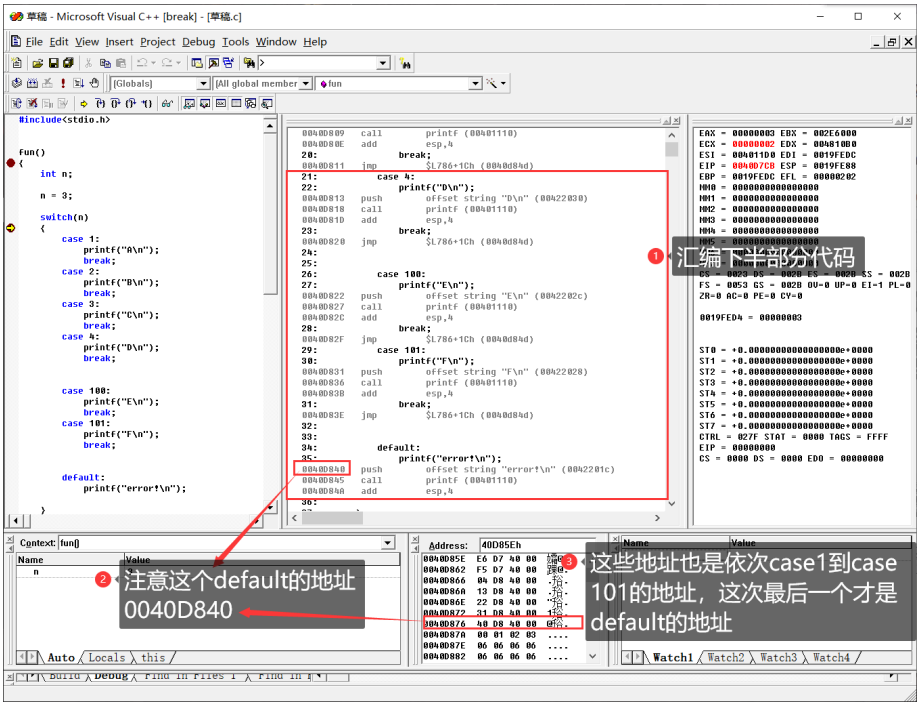
```
14  0040D872  29 D8 40 00  //00 40 D8 29//case 8:入口地址
15  0040D876  CC CC CC CC  烫烫
16  0040D87A  CC CC CC CC  烫烫
```

发现，放`case n:`个数超过三，不连续，但差别较少时，也会生成`大表`，并且将n中间空缺的值 减1 对应edx内存偏移的数据写为default入口地址。

3.情况三：内存中生成`大表`装地址,还会生成`小表`装偏移量。

a.当`case n`不连续，中间n差别很大，两端n连续或差别不大

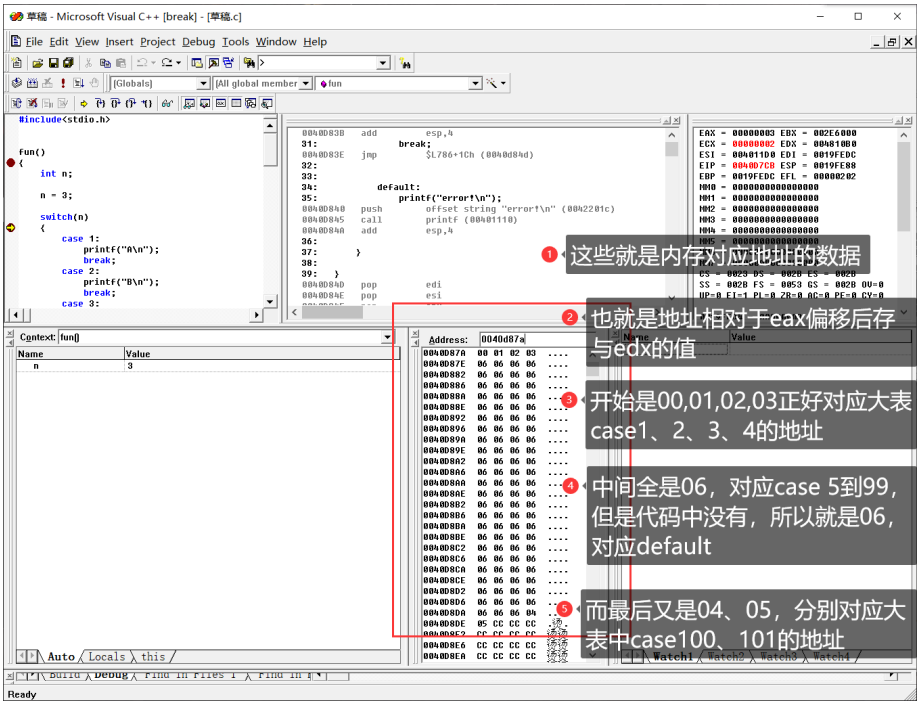




我们把上面那三行不同的汇编写下来

```
0040D7D4    mov         eax,dword ptr [ebp-8]//这个就是n-1的值N
0040D7D7    xor         edx,edx//将edx清零

重点重点重点!
0040D7D9    mov         dl,byte ptr  (0040d87a)[eax]
//这行代码的意思是，将(eax+地址0040d87a)对应地址的 值 存到dl中
//而且这是byet，一个字节的基础，所以只取一个字节的数据
//其值eax的值就是n-1的值。

//所以就相当于这个地址对应eax偏移  取值，也和数组寻址差不多。


0040D7DF    jmp         dword ptr [edx*4+40D85Eh]
//这里40D85Eh同样是大表的位置，
//只不过偏移量edx发生了变化


发现没有，其实0040d87ah这个地址正好在40D85Eh的高位，也就是这两个地址是挨着存数据的!
```

为了探寻edx的变化，我们进入0040d87a这个内存地址。



这些数据就是eax可能的值，你发现没有，大部分是06，而eax为06时，在下面对应`大表`的数据正好是default

所以编译器为我们生成一个`小表`，将`case n:`中空缺的n全部改为改为一个值，这个值对应大表的偏移正好就是default。

这一块内存就是编译器为switch生成的`小表`，用于存储 大表地址的偏移量

而且由于`小表`一个字节，单个字节最大是FF，所以这种情况最多存储255个`case n`

4.总结

大家应该明白为什么要设置switch语句了， 因为当假设情况比较多时，switch明显比if_else`效率`高得多啊。

> *其实switch语句在内存中生成表的过程就是，编译器以内存换效率的过程。*

`case n:`在一定情况下，编译器认为生成表示值得的，应为换了足够多的效率，所以在内存中生成地址表格。但是有时候，每个n都相差巨大，如果还要生成表的话，那需要的内存岂不是很大？所以这是编译器会认为以内存换效率不值得，所以也就不会生成表。

建议自己写vc自己调试一下就立马明白。

所以，我们懂了switch的反汇编后，以后在写switch_case函数时，就要将n的值规范一下，不然也会浪费内存。