

2020/04/02_第4课_VC++ 6.0的使用 if else、switch case语句

笔记本: C

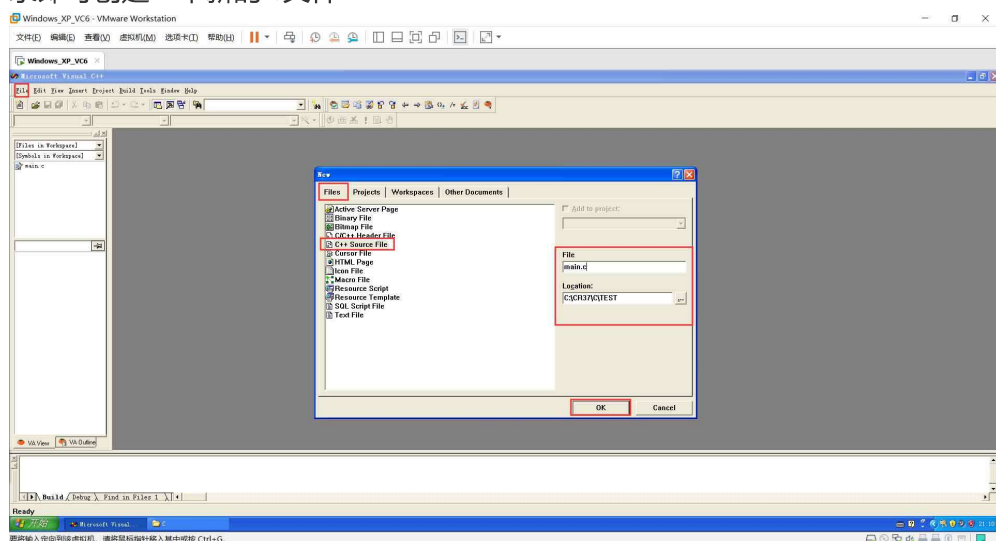
创建时间: 2020/4/3 星期五 9:38

作者: ileemi

- [VC++ 6.0 文件的创建](#)
- [VC++ 6.0 快捷键的使用](#)
 - [操作示例](#)
- [API PK 库函数](#)
- [if else](#)
- [switch case](#)
 - [switch case内部优化方案之一（连续或较为连续方案）](#)
 - [当case值按正顺序排列时在内存中的分布情况如下](#)
 - [当打乱case 值进行逆序排列后，内存中的分布情况如下](#)
 - [将case值进行调大，打乱顺序，case值没有间隔](#)
 - [将case值设定为不连续的，中间进行跳值，case值从20到28，去掉23、25、27](#)
 - [将case值与值之间间隔5个（少5个）](#)
 - [得出结论](#)

VC++ 6.0 文件的创建

- 第一步: 打开Microsoft Visual C++ 6.0
- 第二步: 点击左上角的File --> New
- 第三步: 点击File栏下的C++ Source File, 之后输入对应的文件名及文件存放目录即可创建一个新的.c文件

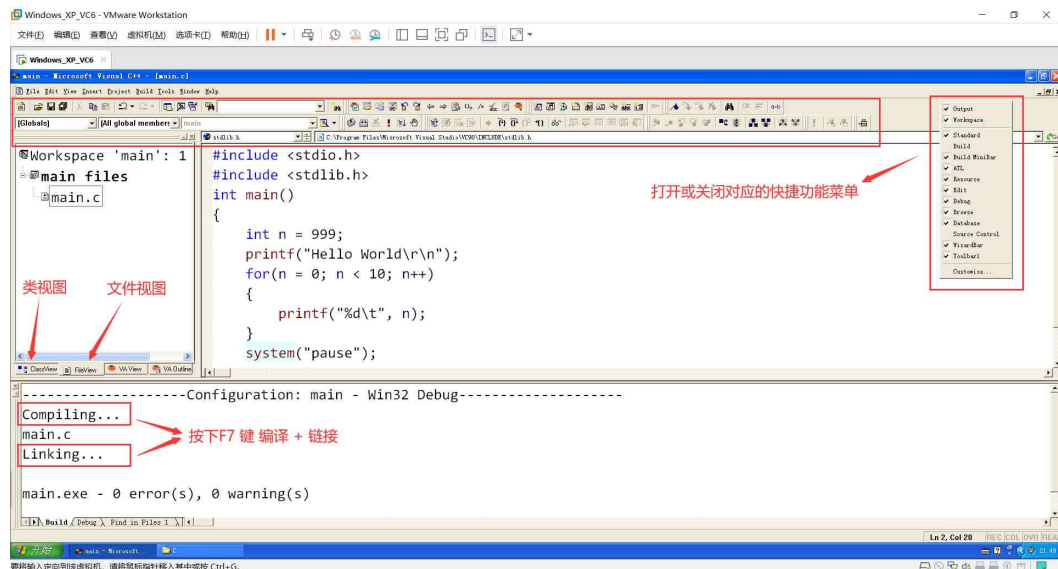


VC++ 6.0 快捷键的使用

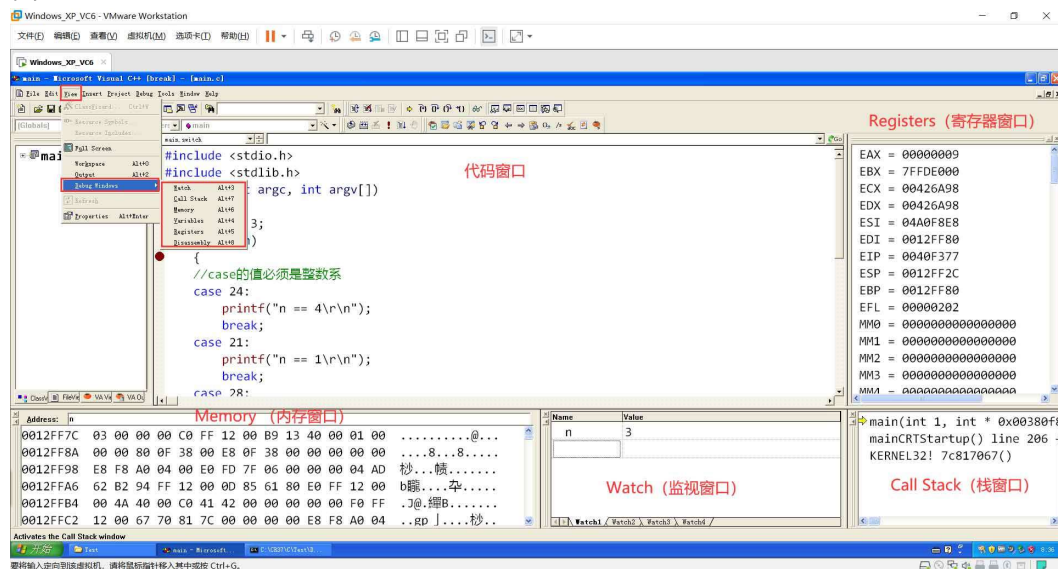
- F1: 调出MSDN (查询函数帮助文档)
- F5: 编译后调试运行
- F7: 编译 + 链接
- F9: 当前行添加/取消断点
- F10: 步过 (单步走)
- F11: 步入 (单步进入, 查看函数实现的功能)
- CTRL + F5: 直接运行
- CTRL + F7: 只编译不链接
- Shift + F5: 停止调试
- Shift + F11: Step out (跳出)
- CTRL + F10: 单步跳转到鼠标点击的光标处
- ALT + 8: 跳转到反汇编代码窗口
- IDE (Integrated Development Environment) : 集成开发环境

操作示例

VC++ 6.0菜单栏的调整:



VC++ 6.0 调试程序时, 需要使用的窗口: 内存窗口、监视窗口、堆栈窗口、寄存器窗口



The screenshot shows a Visual Studio 2005 IDE environment. The main window displays a C program with the following code:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n = 999;
    printf("Hello World\r\n");
    for(n = 0; n < 10; n++)
    {
        printf("%d\t", n);
    }
    system("pause");
}
```

A red annotation "F1 打开对应库函数的帮助文档" points to the `system()` call in the code. The "Library" window is open, showing the "system()" function under "Win32 API". The function's prototype is displayed as `int system(const char *command);`. The "Compatibility" column shows "Win95, Win NT". The "Routines" column lists "system" and "system_s". The "Required Header" column lists "<process.h> or <stdlib.h>". The "Compatibility" column lists "Win95, Win NT". The "Library" window also shows a table of "Libraries" and "Return Value".

Windows XP_CVC6 - VMware Workstation

文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项(O) 帮助(H)

Windows_XP_CVC6

main - Microsoft Visual C++ [break] - [Disassembly]

File Edit View Insert Project Tools Help

Global: [All global members] main

1: Windows
mainCRTStartup, int argc(1)

```

5:      int n = 999;
00400708  C7 45 FC E7 03 00 00  mov     dword ptr [ebp-4],3E7h
6:      printf("Hello World\r\n");
0040070F  68 B0 2F 42 00        push   offset string "Hello World\r\n" (00422fb0)
00400714  E8 47 39 FF FF        call  01000000
00400719  83 C4 04              add     eax,4
7:      for(n = 0; n < 10; n++)
0040071C  C7 45 FC 00 00 00 00  mov     dword ptr [ebp-4],0
00400723  EB 09                jmp     00400727
00400725  8B 45 FC              mov     eax,dword ptr [ebp-4]

```

反汇编代码窗口

Go To Source
Show Next Statement
Insert/Remove Breakpoint
Step Into
Step Over
Step Out
Set Next Statement
Set Breakpoint
Source Annotation
Code Bytes
Disassembly View

EAX = CCCCCCCC
EBX = 7FFD6000
ECX = 00000000
EDX = 00380F8E
ESI = 058FF8E8
EDI = 0012FF80
EIP = 00400708
ESP = 0012FF30
EBP = 0012FF80
EFL = 00002116

Context: [main(int, int*)]

Name	Value
argc	1
argv	0x00380f80
n	-858993460

Loaded 'C:\WINDOWS\system32\kernel32.dll', no matching symbolic information found.

Build, Debug (Find in Files) |

Activates the source window for this instruction

main - Microsoft

源程序输入定向到该虚拟机。 请将鼠标指针移至其中或按 Ctrl+G。

The screenshot shows a Windows XP virtual machine running Visual Studio. The main window displays a C++ program that prints "Hello World" and a loop counter 'n' from 0 to 9. The program is compiled and run, and the output is visible in the console. The memory dump at the bottom shows the stack frame for main, with the variable 'n' at address 00010000 containing the value 0. The console output shows "Hello World" and the loop counter values.

Code in main.c:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n = 999;
    printf("Hello World\r\n");
    for(n = 0; n < 10; n++)
    {
        printf("%d\t", n);
    }
}
```

Memory dump (Address: 00010000):

Address	Disassembly	Comment
0000FFDC	?? ?? ?? ?? ?? ?? ?? ?? ?? ??	??????????????
0000FFE8	?? ?? ?? ?? ?? ?? ?? ?? ?? ??	??????????????
0000FFF4	?? ?? ?? ?? ?? ?? ?? ?? ?? ??	??????????????
00010000	3D 00 3A 00 3A 00 3D 00 3A 00 3A 00	=...=...
0001000C	5C 00 00 00 3D 00 43 0A 00 3D 00	...\..C...\.

Register values (EAX, EBX, ECX, EDX, ESI, EDI, EIP, ESP, EBP, EFL):

- EAX = 0000000D
- EBX = 7FFD7000
- ECX = 00424A60
- EDX = 00424A60
- ESI = 03C4F8E8
- EDI = 0012FF80
- EIP = 0040D734
- ESP = 0012FF30
- EBP = 0012FF80
- EFL = 0000297

Stack frame (Context: [main]):

Name	Value
n	0

Console output:

```
Hello World
0
1
2
3
4
5
6
7
8
9
```

Loaded 'C:\WINDOWS\system32\kernel32.dll', no matching symbolic information found.

API PK 库函数

看环境：如果没有造缓存， fread 就比 库函数的执行效率要慢一点。

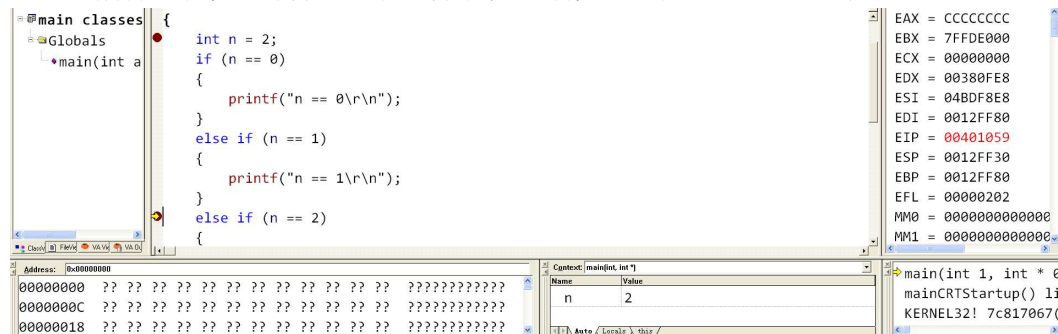
malloc (C库函数) --> HeapAlloc (API函数)

不同平台的C编译器会对这个平台接口做一定的封装工作。

if else

可以做复杂的条件判断、可以设定条件的优先级（将执行数据量大的条件放置在前）、可做区间判断、使用灵活。

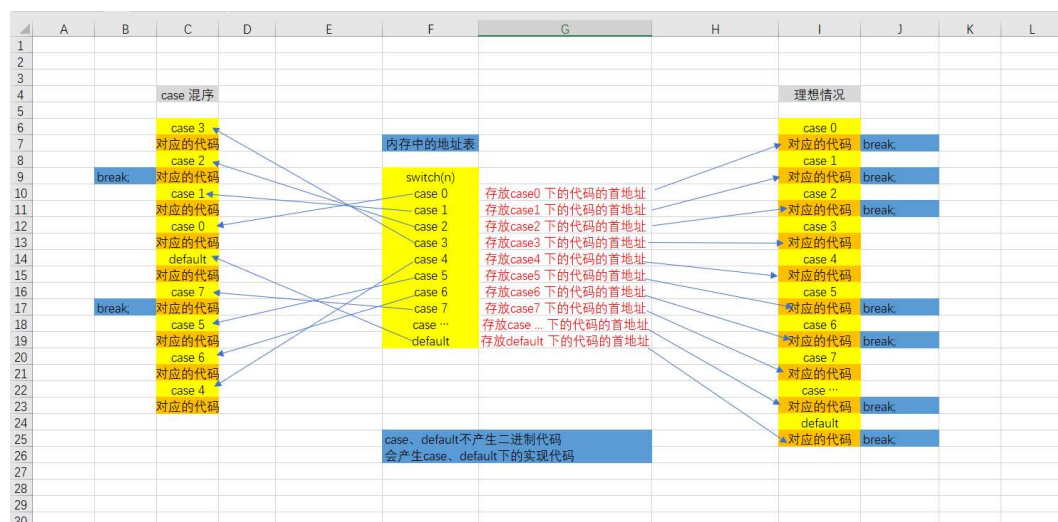
缺点：阶梯比较，如果数量比较次数较多的话，就没有switch case效果好。



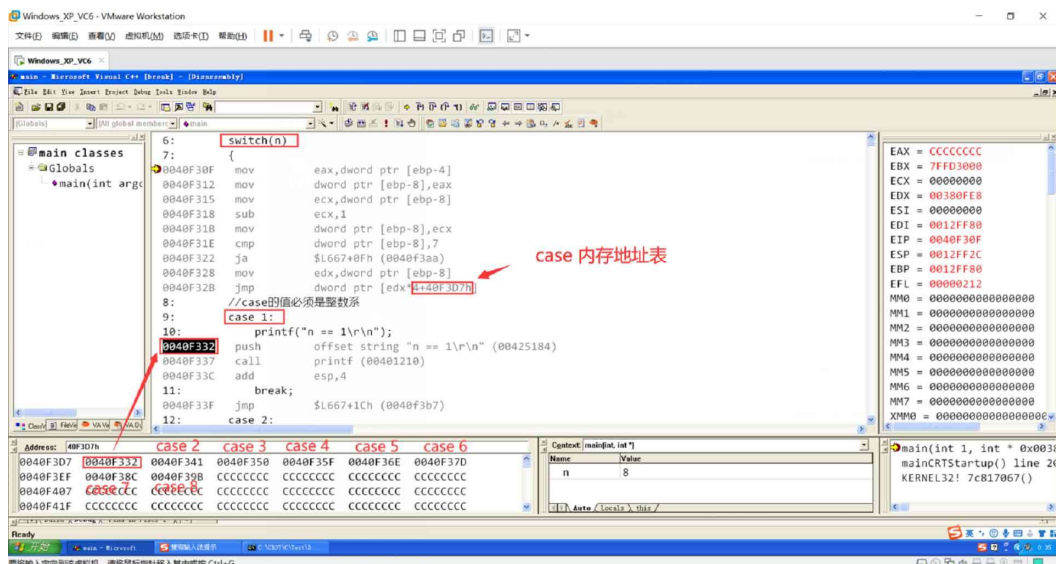
switch case

case 的值必须是整型系：整数或者类整型 ASCII ('A')， long (5L) 型都可以。switch语句是一个优化版的分支结构，在汇编的年代这个结构很多人在用，后来C/C++标准委员会觉得这个结构很实用，之后就将其集成到C语言的语法框架里，之后将其命名为switch结构。俩次到位，第一次求得switch表达式中的值，根据值访问内存中对应的地址表进行跳转。

switch case内部优化方案之一（连续或较为连续方案）



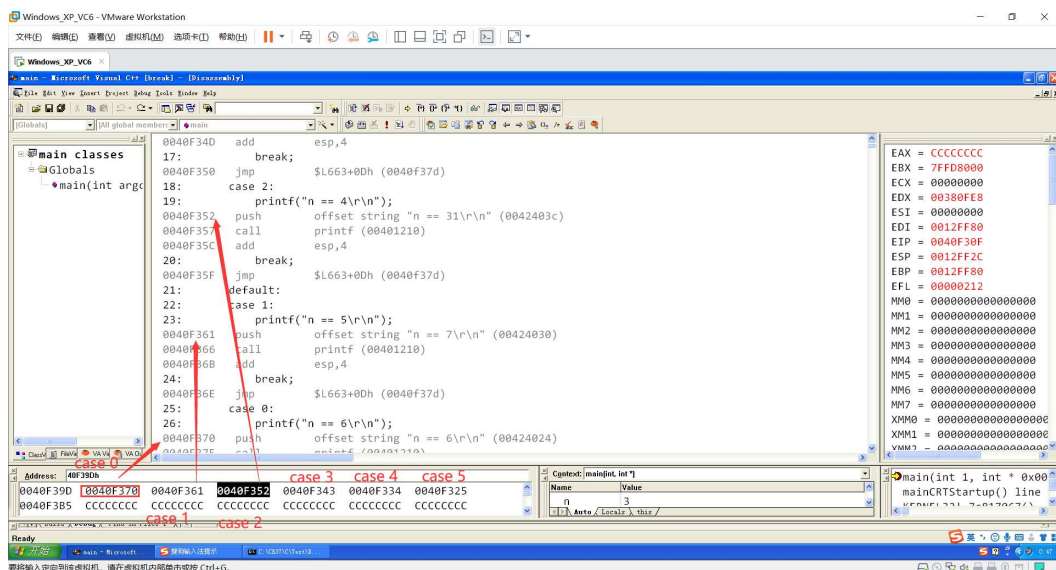
当case值按正顺序排列时在内存中的分布情况如下



发现在内存地址表中，存储的case值下对应的代码地址是连续的

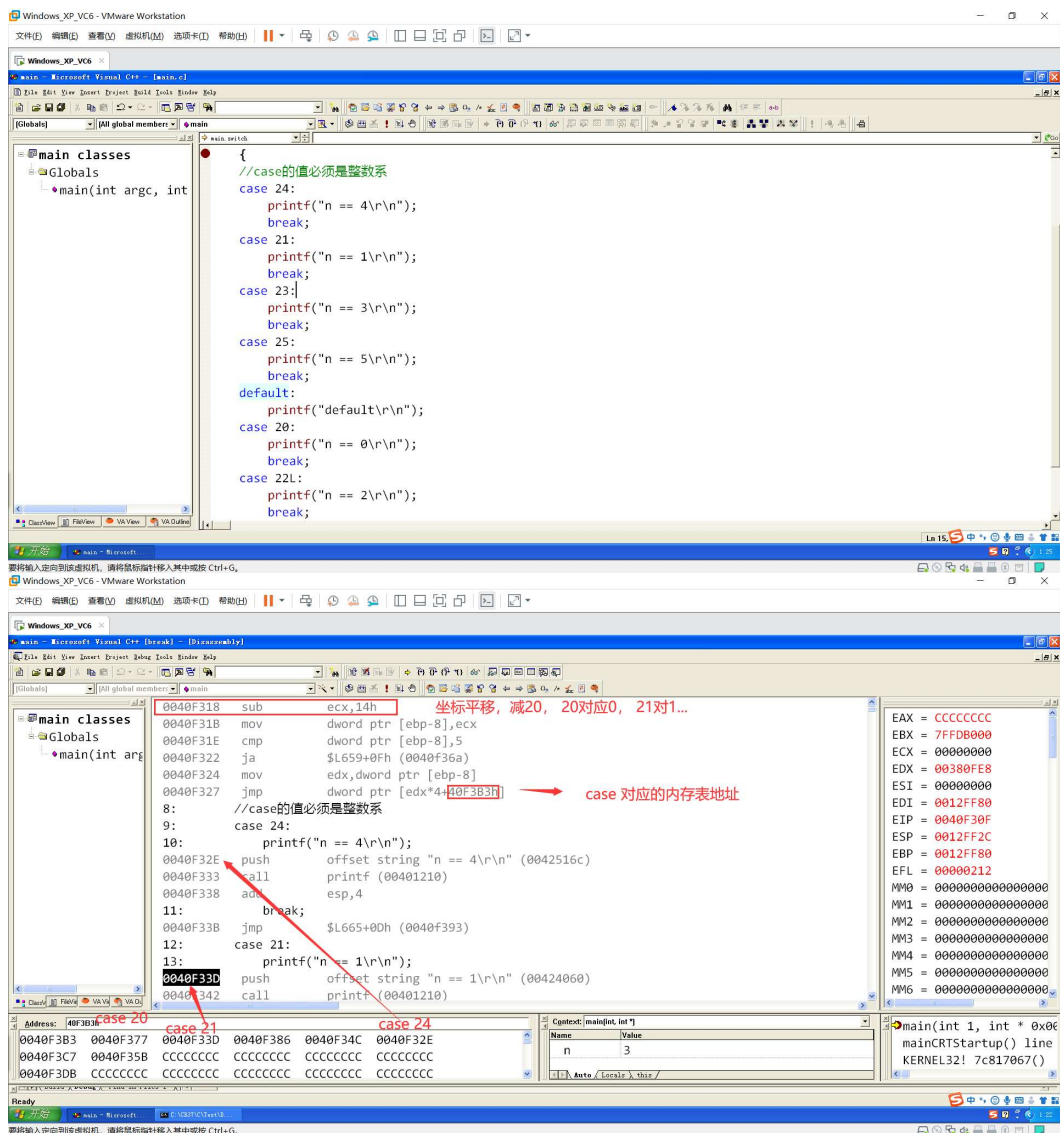
基本结构思想：根据switch中表达式的值，找到内存中case内存地址表的对应的下标值。

当打乱case 值进行逆序排列后，内存中的分布情况如下



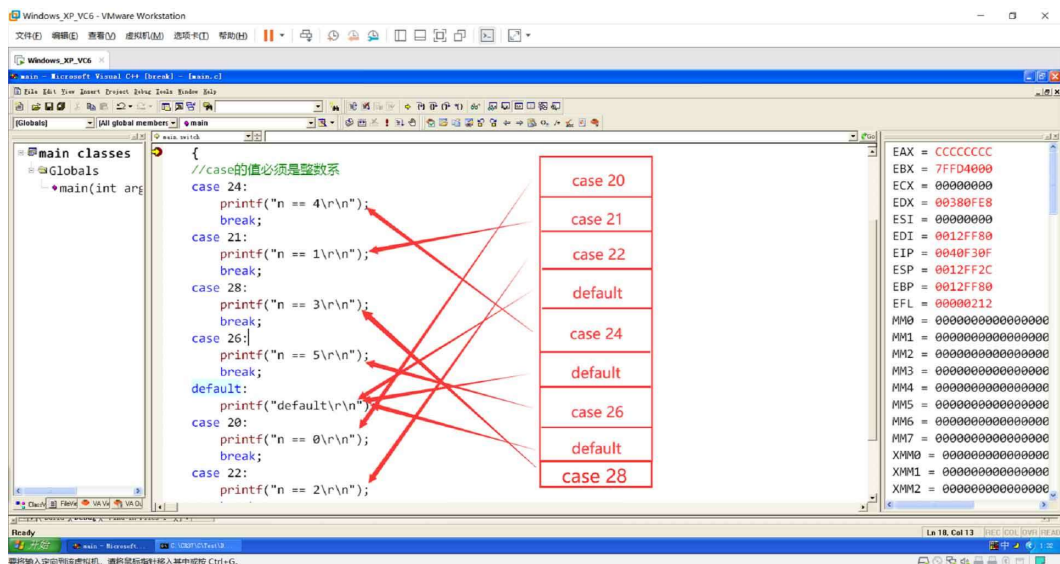
即使case 后的值顺序进行了更改，但是通过case 的内存地址表可以发现，case值依然是顺序排列的。

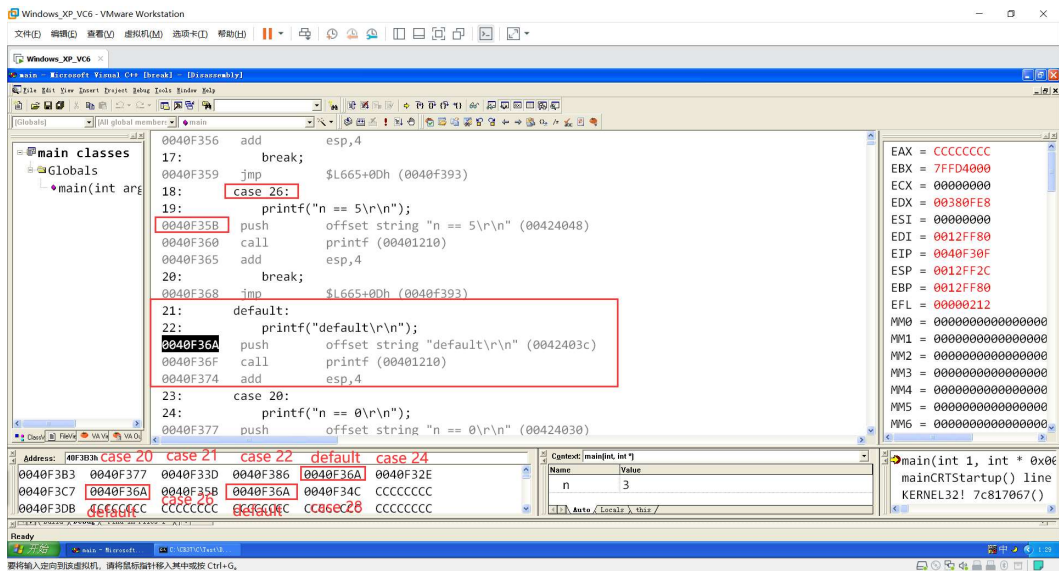
将case值进行调大，打乱顺序，case值没有间隔



通过反汇编窗口可以发现case值从20开始，前20是空值，就像数组中的前20元素是空值一样，存储空值是没有意义的，所以系统内部进行了坐标平移，即使case值顺序打乱后，通过case 内存地址表可以发现，其值依然是按循序进行排列的

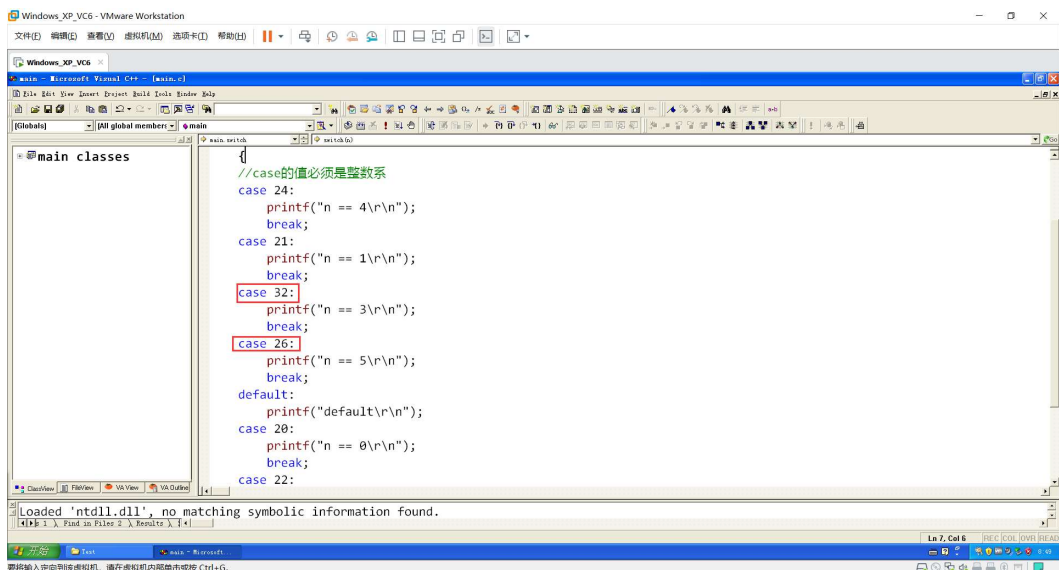
将case值设定为不连续的，中间进行跳值，case值从20到28，去掉23、25、27



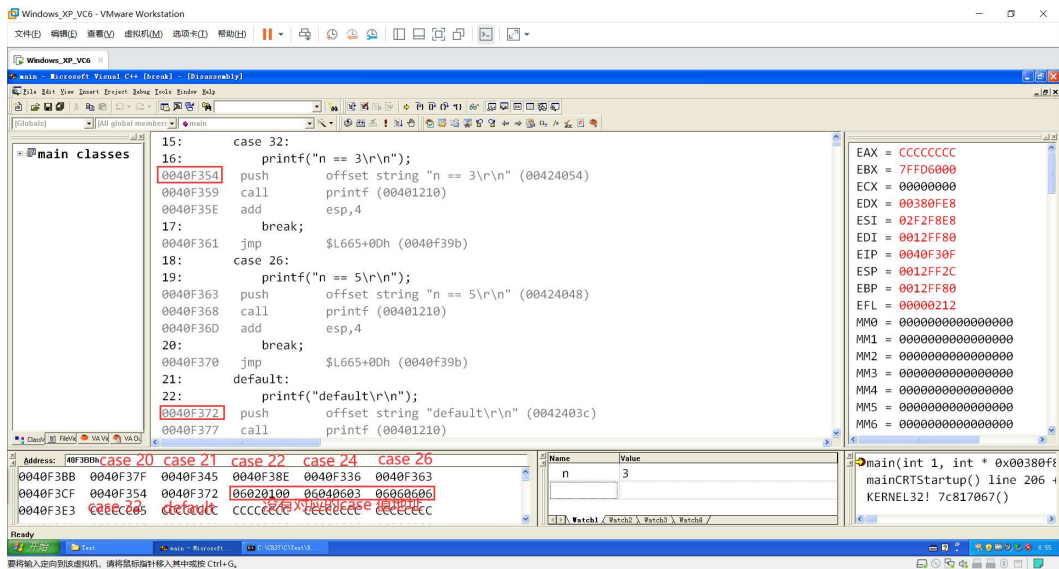


通过case 地址内存表的地址以及反汇编窗口可以发现，其case值依旧是连续排列的，中间缺少的值case 23、case 25、case27都用default下的语句地址进行了填充。

将case值与值之间间隔5个（少5个）



接着调试程序，Alt + 8 打开反汇编窗口查看case值对应的地址排列情况



可以看出case值在内存中的分布已经不在连续了

得出结论

- 每个switch case分支的访问效率均等
- 不管怎样调整case的位置都不会影响它们的访问效率

switch case 结构有四套优化方案，上面演示的就是连续或较为连续的方案：case的数量需要大于三个，只有这样才有优化行为，case 的值需要连续，如果不连续，中间欠缺的值不能超过5个，如果超过5个或者等于5个，那么其在内存中就不是连续存储的。

case 的值不能为小数是因为其在内存中存储的结构类似于数组的存储结构，需要根据switch表达式的值在内存中进行查表，基于此，case 的值必须是类整型。