函数跳转表

在**Debug**下，不开随机基址时，函数跳转表为 0x00401000 处，存放着用户定义的函数的跳转表，在随机基址下，模块基址加 0x00001000

```
00401001 CC                   int         3
00401002 CC                   int         3
00401003 CC                   int         3
00401004 CC                   int         3
@ILT+0(_sum):
00401005 E9 B6 00 00 00       jmp         main+20h (004010c0)
@ILT+5(_main):
0040100A E9 91 00 00 00       jmp         main (004010a0)
0040100F E9 0C 00 00 00       jmp         foo1 (00401020)
00401014 E9 47 00 00 00       jmp         foo2 (00401060)
00401019 CC                   int         3
0040101A CC                   int         3
0040101B CC                   int         3
0040101C CC                   int         3
0040101D CC                   int         3
0040101E CC                   int         3
0040101F CC                   int         3
```

高版本下代码量很大的情况下，可能会出现不是用户定义也不是库函数的函数在此表中

```
___acrt_thread_attach:
00411212 E9 A9 3C 00 00       jmp         __scrt_stub_for_acrt_thread_attach (0414EC0h)
@__CheckForDebuggerJustMyCode@4:
00411217 E9 E4 07 00 00       jmp         __CheckForDebuggerJustMyCode (0411A00h)
_atexit:
0041121C E9 BF 20 00 00       jmp         atexit (04132E0h)
__RTC_CheckEsp:
00411221 E9 0A 0A 00 00       jmp         _RTC_CheckEsp (0411C30h)
___scrt_dllmain_exception_filter:
00411226 E9 85 1B 00 00       jmp         __scrt_dllmain_exception_filter (0412DB0h)
__onexit:
0041122B E9 C0 1F 00 00       jmp         _onexit (04131F0h)
___scrt_stub_for_is_c_termination_complete:
00411230 E9 CB 3C 00 00       jmp         __scrt_stub_for_is_c_termination_complete (0414F00h)
__scrt_narrow_environment_policy::initialize_environment:
00411235 E9 A6 0E 00 00       jmp         __scrt_narrow_environment_policy::initialize_environment (04120E0h)
_QueryPerformanceCounter@4:
0041123A E9 FF 3B 00 00       jmp         _QueryPerformanceCounter@4 (0414E3Eh)
_foo1:
0041123F E9 DC 05 00 00       jmp         foo1 (0411820h)    已用时间<=1ms
___scrt_get_dyn_tls_dtor_callback:
00411244 E9 A7 23 00 00       jmp         __scrt_get_dyn_tls_dtor_callback (04135F0h)
@_RTC_CheckStackVars@8:
00411249 E9 62 09 00 00       jmp         _RTC_CheckStackVars (0411BB0h)
___scrt_uninitialize_crt:
0041124E E9 4D 1F 00 00       jmp         __scrt_uninitialize_crt (04131A0h)
_FreeLibrary@4:
00411253 E9 3A 3C 00 00       jmp         _FreeLibrary@4 (0414E92h)
__CRT_RTC_INITW:
00411258 E9 23 0A 00 00       jmp         _CRT_RTC_INITW (0411C80h)
__crt_at_quick_exit:
0041125D E9 9A 3B 00 00       jmp         __crt_at_quick_exit (0414DFCh)
_exit:
00411262 E9 23 3B 00 00       jmp         _exit (0414D8Ah)
__initialize_default_precision:
00411267 E9 94 22 00 00       jmp         _initialize_default_precision (0413500h)
___acrt_thread_detach:
0041126C E9 5F 3C 00 00       jmp         __scrt_stub_for_acrt_thread_detach (0414ED0h)
```

指针

函数指针

数组名为数组的首地址，那么函数名也就为函数的首地址，即函数中第一条指令的地址

```
返回值 (调用约定 *指针名)(参数列表);

// 例如:
int __cdelc foo(int a, double b);
int (__cdelc *foo_ptr)(int, double) = foo;

foo(1, 10.0);
(*foo_ptr)(1, 10.0);     // 函数指针间接调用
foo_ptr(1, 10.0);        // 函数指针省略写法
```

注：函数指针无法做运算

函数指针的应用

1. 可用在任意想要执行代码的地方插入代码执行（热补丁、栈上执行恶意代码等）
2. 可用在中间件上组织函数调用
   以上在作业中有所体现