C++反汇编逆向分析2（学习笔记）

看雪论坛ID：zhang_derek

看雪论坛原文地址：https://bbs.kanxue.com/thread-275320.htm

一、准备工作

1.1.编译环境

- VS2019
- Release/Debug x86

1.2.OllyDbg

常用快捷键

- F2：设置断点
- F3：加载一个可执行程序
- F4：程序执行到光标处
- F5：缩小，还原当前窗口
- F7：单步步入
- F8：单步步过
- F9：运行程序
- Ctrl+F2：重新运行程序到起始处
- Ctrl+F9：执行到函数返回处，用于跳出函数实现
- Alt+F9：执行到用户代码处，用于快速跳出系统函数
- Ctrl+G：快速定位跳转地址

1.3.IDA

常用快捷键

- 空格键：反汇编窗口切换文本跟图形
- a：解析成字符串的首地址
- b：十六进制与二进制转换
- c：解释位一条指令
- d：解释为数据，每按一次转换数据长度
- g：快速查找到对应地址
- h：十六进制与十进制转换
- k：将数据解释为栈变量
- m：解释为枚举成员
- n：重新命名
- t：把偏移改为结构体
- u：取消定义函数、代码、数据的定义
- x：查看交叉引用
- y：更改变量的类型
- 分号：添加注释
- shift+F9：添加结构体
- Alt+T：搜索文本
- ins：插入结构体
- Alt+Q：修改数据类型为结构体类型

二、基本数据类型的表现形式

2.1.无符号整数

以unsigned int为例

- 取值范围：0～4294967295（0x00000000～0xFFFFFFFF）
- 小尾方式存放：低数据位存放在内存的低端，高数据位存放在内存的高端
- 不存在正负之分，都是正数

2.2.有符号正数

以int为例

- 最高位是符号位，0表示正数，1表示负数
- 取值范围：-2147483648~2147483648
- 正数区间（0x00000000～0x7FFFFFFF），负数区间（0x80000000～0xFFFFFFFF）
- 负数在内存中都是以补码形式存放的，可以表达位：对这个数值取反+1

2.3.浮点数类型

SSE指令集

- 八个寄存器：XMM0-XMM8，每个寄存器占16字节（128bit）

指令名称	使用格式	指令功能
MOVSS	xmm1,xmm2 xmm1,mem32 xmm2/mem32,xmm1	传送单精度数
MOVSD	xmm1,xmm2 xmm1,mem64 xmm2/mem64,xmm1	传送双精度数
MOVAPS	xmm1,xmm2/mem128 xmm1/mem128, xmm2	传送对齐的封装好的单精度数
MOVAPD	xmm1,xmm2/mem128 xmm1/mem128, xmm2	传送对齐的封装好的双精度数
ADDSS	xmm1, xmm2/mem32	单精度数加法
ADDSD	xmm1, xmm2/mem64	双精度数加法
ADDPS	xmm1, xmm2/mem128	并行 4 个单精度数加法
ADDPD	xmm1, xmm2/mem128	并行 2 个双精度数加法
SUBSS	xmm1, xmm2/mem32	单精度数减法
SUBSD	xmm1, xmm2/mem64	双精度数减法
SUBPS	xmm1, xmm2/mem128	并行 4 个单精度数减法
SUBPD	xmm1, xmm2/mem128	并行 2 个双精度数减法
MULSS	xmm1, xmm2/mem32	单精度数乘法
MULSD	xmm1, xmm2/mem64	双精度数乘法
MULPS	xmm1, xmm2/mem128	并行 4 个单精度数乘法
MULPD	xmm1, xmm2/mem128	并行 2 个双精度数乘法
DIVSS	xmm1, xmm2/mem32	单精度数除法
DIVSD	xmm1, xmm2/mem64	双精度数除法
DIVPS	xmm1, xmm2/mem128	并行 4 个单精度数除法
DIVPD	xmm1, xmm2/mem128	并行 2 个双精度数除法
CVTTSS2SI	reg32,xmm1/mem32 reg64,xmm1/mem64	用截断的方法将单精度数转换为整数
CVTTSD2SI	reg32,xmm1/mem64 reg64,xmm1/mem64	用截断的方法将双精度数转换为整数
CVTSI2SS	xmm1,reg32/mem32 xmm1,reg64/mem64	将整数转换为单精度数
CVTSI2SD	xmm1,reg32/mem32 xmm1,reg64/mem64	将整数转换为双精度数

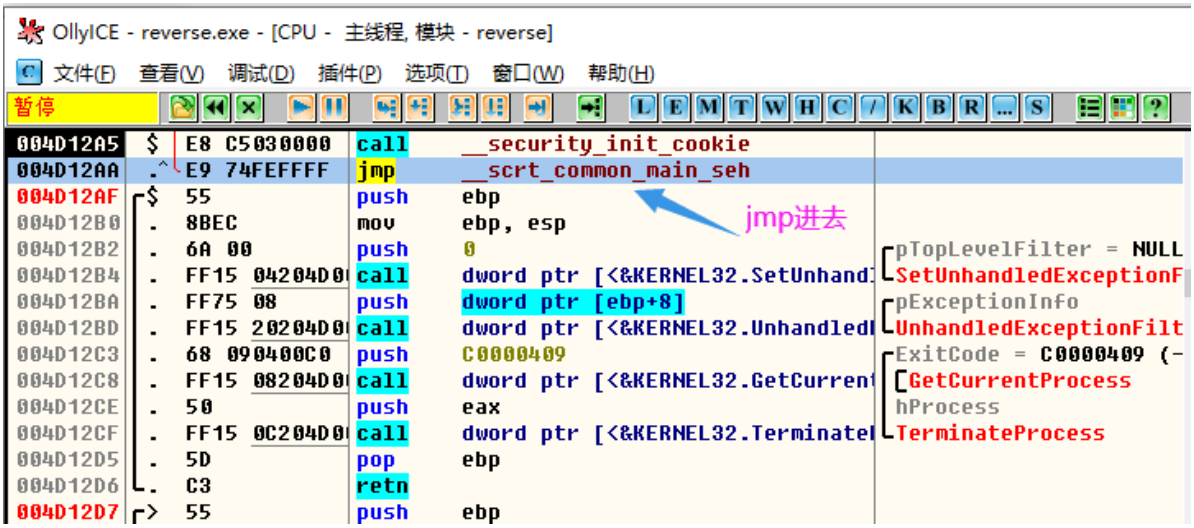
三、认识启动函数，找用户入口

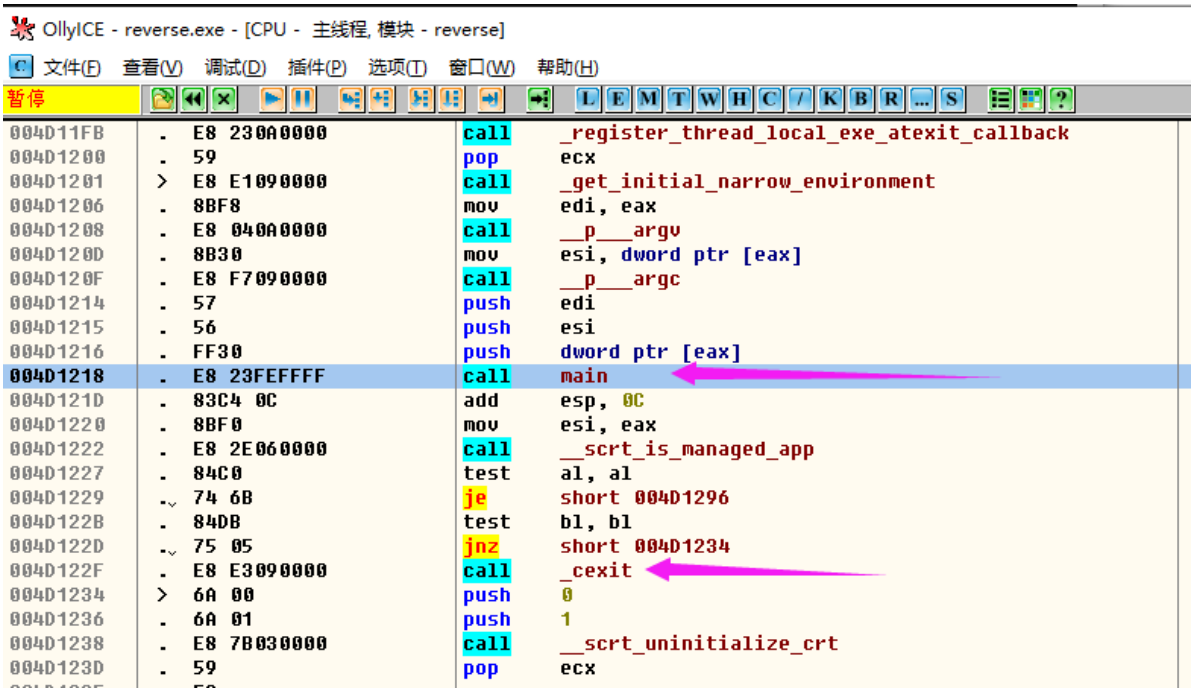
3.1.找main函数入口方法

VS2019 Release版本

- 有3个参数，main函数是启动函数中唯一具有3个参数的函数
- 找到入口代码第一次调用exit函数处，离exit最近的且有3个参数的函数通常就是main函数

找main函数入口





四、观察各种表达式的求值过程

4.1.加法

release版

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int n1 = argc;
    int n2 = argc;        //复写传播：n2等价于引用argc， n2则被删除
    n1 = n1 + 1;          //下一句n1重新赋值了，所以这句被删除了
    n1 = 1 + 2;          //常量折叠：n1 = 3
    n1 = n1 + n2;        //常量传播和复写传播： n1 = 3 + argc
    printf("n1 = %d\n", n1);
    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 mov     eax, [ebp+argc] ; n2 = argc
.text:00401046                 add     eax, 3          ; n1 = n1 + n2 = 3 + argc
.text:00401049                 push    eax
.text:0040104A                 push    offset _Format  ; "n1 = %d\n"
.text:0040104F                 call    _printf
.text:00401054                 add     esp, 8
.text:00401057                 xor     eax, eax
.text:00401059                 pop     ebp
.text:0040105A                 retn
.text:0040105A _main           endp
.text:0040105A
```

4.2.减法

release版

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int n1 = argc;
    int n2 = 0;
    scanf_s("%d", &n2);
    n1 = n1 - 100;
    n1 = n1 + 5 - n2;        //n1 = n1 -95 - n2
    printf("n1 = %d \r\n", n1);
    return 0;
}
```

ida

```
.text:00401090 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401090 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401090
.text:00401090 n2              = dword ptr -8
.text:00401090 var_4           = dword ptr -4
.text:00401090 argc            = dword ptr  8
.text:00401090 argv            = dword ptr  0Ch
.text:00401090 envp            = dword ptr  10h
.text:00401090
.text:00401090                 push    ebp
.text:00401091                 mov     ebp, esp
.text:00401093                 sub     esp, 8
.text:00401096                 mov     eax, ___security_cookie
.text:0040109B                 xor     eax, ebp
.text:0040109D                 mov     [ebp+var_4], eax
.text:004010A0                 lea     eax, [ebp+n2]
.text:004010A3                 mov     [ebp+n2], 0
.text:004010AA                 push    eax             ; &n2
.text:004010AB                 push    offset _Format  ; "%d"
.text:004010B0                 call    _scanf_s
.text:004010B5                 mov     eax, [ebp+argc] ; n1 = argc
.text:004010B8                 sub     eax, [ebp+n2]   ; n1 = n1 - n2
.text:004010BB                 sub     eax, 95         ; n1 = n1 - 95
.text:004010BE                 push    eax
.text:004010BF                 push    offset aN1D     ; "n1 = %d \r\n"
.text:004010C4                 call    _printf
.text:004010C9                 mov     ecx, [ebp+var_4]
.text:004010CC                 add     esp, 10h
.text:004010CF                 xor     ecx, ebp        ; StackCookie
.text:004010D1                 xor     eax, eax
.text:004010D3                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:004010D8                 mov     esp, ebp
.text:004010DA                 pop     ebp
.text:004010DB                 retn
.text:004010DB _main           endp
```

4.3.乘法

release版

```c
#include <stdio.h>
int main(int argc, char* argv[]) {
    int n1 = argc;
    int n2 = argc;

    printf("n1 * 15 = %d\n", n1 * 15);        //变量乘常量 ( 常量值为非 2 的幂 )
    printf("n1 * 16 = %d\n", n1 * 16);        //变量乘常量 ( 常量值为 2 的幂 )
    printf("2 * 2 = %d\n", 2 * 2);            //两常量相乘
    printf("n2 * 4 + 5 = %d\n", n2 * 4 + 5); //混合运算
    printf("n1 * n2 = %d\n", n1 * n2);        //两变量相乘
    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 push    esi
.text:00401044                 mov     esi, [ebp+argc]
.text:00401047                 mov     eax, esi
.text:00401049                 shl     eax, 4          ; esi x 16
.text:0040104C                 sub     eax, esi        ; esi x 16 - esi = esi x 15
.text:0040104E                 push    eax
.text:0040104F                 push    offset _Format  ; "n1 * 15 = %d\n"
.text:00401054                 call    _printf
.text:00401059                 mov     eax, esi
.text:0040105B                 shl     eax, 4          ; esi x 16
.text:0040105E                 push    eax
.text:0040105F                 push    offset aN116D   ; "n1 * 16 = %d\n"
.text:00401064                 call    _printf
.text:00401069                 push    4
.text:0040106B                 push    offset a22D     ; "2 * 2 = %d\n"
.text:00401070                 call    _printf
.text:00401075                 lea     eax, ds:5[esi*4] ; esi x 4 + 5
.text:0040107C                 push    eax
.text:0040107D                 push    offset aN245D   ; "n2 * 4 + 5 = %d\n"
.text:00401082                 call    _printf
.text:00401087                 imul    esi, esi
.text:0040108A                 push    esi
.text:0040108B                 push    offset aN1N2D   ; "n1 * n2 = %d\n"
.text:00401090                 call    _printf
.text:00401095                 add     esp, 28h
.text:00401098                 xor     eax, eax
.text:0040109A                 pop     esi
.text:0040109B                 pop     ebp
.text:0040109C                 retn
.text:0040109C _main           endp
.text:0040109C
```

4.4.除法

常用指令

- cdq：把eax的最高位填充到edx，如果eax ≥ 0，edx = 0，如果eax < 0，edx = 0xFFFFFFFF
- sar：算术右移
- shr：逻辑右移
- neg：将操作数取反+1
- div：无符号数除法
- idiv：有符号除法
- mul：无符号数乘法
- imul：有符号数乘法

4.4.1.除数为无符号2的幂

release版

```
#include <stdio.h>
int main(unsigned argc, char* argv[]) {

    printf("a / 16 = %u", argc / 16);

    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 mov     eax, [ebp+argc]
.text:00401046                 shr     eax, 4          ; argc / 16
.text:00401049                 push    eax
.text:0040104A                 push    offset _Format  ; "argc / 16 = %u"
.text:0040104F                 call    _printf
.text:00401054                 add     esp, 8
.text:00401057                 xor     eax, eax
.text:00401059                 pop     ebp
.text:0040105A                 retn
.text:0040105A _main           endp
```

4.4.2.除数为无符号非2的幂

release版

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    printf("argc / 3 = %u", (unsigned)argc / 3); //变量除以常量，常量为无符号非2的幂
    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 mov     eax, 0AAAAAAABh ; M = 2863311531
.text:00401048                 mul     [ebp+argc]      ; 无符号数乘法， edx.eax = argc * M
.text:0040104B                 shr     edx, 1          ; edx = argc * M >> 32 >> 1
.text:0040104D                 push    edx
.text:0040104E                 push    offset _Format  ; "argc / 3 = %u"
.text:00401053                 call    _printf
.text:00401058                 add     esp, 8
.text:0040105B                 xor     eax, eax
.text:0040105D                 pop     ebp
.text:0040105E                 retn
.text:0040105E _main           endp
```

总结

- c = 2^n / M = 2^{32+1} / 2863311531 = 2.9999 = 3
- 其中n为右移的次数

4.4.3.另一种除数为无符号非2的幂

release版

```
#include <stdio.h>
int main(unsigned argc, char* argv[]) {

    printf("a / 7 = %u", argc / 7);

    return 0;
}
```

ida反汇编

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 mov     ecx, [ebp+argc] ; ecx = argc
.text:00401046                 mov     eax, 613566757  ; eax = M
.text:0040104B                 mul     ecx             ; edx.eax = argc*M
.text:0040104D                 sub     ecx, edx        ; ecx = argc-(argc*M >> 32)
.text:0040104F                 shr     ecx, 1          ; ecx = (argc-(argc*M >> 32)) >>1
.text:00401051                 add     ecx, edx        ; ecx = ((argc-(argc*M >> 32))>>1) + (argc*M>>32)
.text:00401053                 shr     ecx, 2          ; ecx=ecx = (((argc-(argc*M >> 32))>>1) + (argc*M>>32))>>2
.text:00401056                 push    ecx
.text:00401057                 push    offset _Format  ; "a / 16 = %u"
.text:0040105C                 call    _printf
.text:00401061                 add     esp, 8
.text:00401064                 xor     eax, eax
.text:00401066                 pop     ebp
.text:00401067                 retn
.text:00401067 _main           endp
```

总结:

$$c = \frac{2^{32+n}}{2^{32} + M}$$

$2^{32+1+2} = 2^{35} = 34,359,738,368$

$2^{32} + 613566757 = 4,908,534,053$

结果 = 34,359,738,368 / 4,908,534,053 = 6.9999， 然后向上取整 = 7

4.4.4.除数为有符号2的幂

release版

```
#include <stdio.h>
int main(int  argc, char* argv[]) {

    printf("a / 8 = %d", argc / 8);

    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 mov     eax, [ebp+argc]
.text:00401046                 cdq                     ; if eax >= 0, edx=0, else edx = 0xFFFFFFFF
.text:00401047                 and     edx, 7          ; if eax >= 0, edx=0, else edx = 7
.text:0040104A                 add     eax, edx        ; if eax >= 0, eax = eax + 0, else eax = eax + 7
.text:0040104C                 sar     eax, 3          ; eax >> 3,相当于除以8
.text:0040104F                 push    eax
.text:00401050                 push    offset _Format  ; "a / 8 = %u"
.text:00401055                 call    _printf
.text:0040105A                 add     esp, 8
.text:0040105D                 xor     eax, eax
.text:0040105F                 pop     ebp
.text:00401060                 retn
.text:00401060 _main           endp
.text:00401060
```

总结：除数 = 2^n ， n的数值为右移了多少位

4.4.5.除数为有符号非2的幂

release版

```
#include <stdio.h>
int main(int  argc, char* argv[]) {

    printf("a / 9 = %d", argc / 9);    ////变量除以常量，常量为非2的幂

    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 mov     eax, 954437177  ; eax = M
.text:00401048                 imul    [ebp+argc]      ; edx.eax=argc*M
.text:0040104B                 sar     edx, 1          ; edx=(argc*M>>32)>>1
.text:0040104D                 mov     eax, edx
.text:0040104F                 shr     eax, 31         ; eax=eax>>31取符号位
.text:00401052                 add     eax, edx        ; if(edx < 0)，eax=((argc*M>>32)>>1)+1
.text:00401052                                         ; ;if(edx >=0)，eax=(argc*M>>32)>>1
.text:00401054                 push    eax
.text:00401055                 push    offset _Format  ; "a / 9 = %d"
.text:0040105A                 call    _printf
.text:0040105F                 add     esp, 8
.text:00401062                 xor     eax, eax
.text:00401064                 pop     ebp
.text:00401065                 retn
.text:00401065 _main           endp
```

总结：除数 = 2^{32+1} / 954437177 = 8.9999 = 9

4.4.6.第二种除数为有符号非2的幂

release版

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    printf("argc / 7 = %d", argc / 7); //变量除以常量，常量为非2的幂
    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 mov     eax, 92492493h  ; M = 2454267027
.text:00401048                 imul    [ebp+argc]      ; edx.eax = argc*M
.text:0040104B                 add     edx, [ebp+argc] ; edx = (argc*M >> 32) + argc
.text:0040104E                 sar     edx, 2          ; edx = ((argc*M >> 32) + argc) >> 2
.text:00401051                 mov     eax, edx        ; eax = edx
.text:00401053                 shr     eax, 31         ; eax = eax >> 31 取符号位
.text:00401056                 add     eax, edx
.text:00401058                 push    eax
.text:00401059                 push    offset _Format  ; "argc / 7 = %d"
.text:0040105E                 call    _printf
.text:00401063                 add     esp, 8
.text:00401066                 xor     eax, eax
.text:00401068                 pop     ebp
.text:00401069                 retn
.text:00401069 _main           endp
```

总结：除数 = 2^{32+2} / 2454267027 = 6.9999 = 7

4.4.7.除数为有符号负2的幂

release版

```
#include <stdio.h>
int main(int  argc, char* argv[]) {

    printf("a / -4 = %d", argc / -4);

    return 0;
}
```

ida反汇编

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 mov     eax, [ebp+argc]
.text:00401046                 cdq                     ; if eax >> 0 ,edx = 0, else edx = 0xffffffff
.text:00401047                 and     edx, 3          ; if eax >> 0 ,eax = eax+0, else eax = eax+3
.text:0040104A                 add     eax, edx
.text:0040104C                 sar     eax, 2
.text:0040104F                 neg     eax             ; eax = -eax
.text:00401051                 push    eax
.text:00401052                 push    offset _Format  ; "a / -4 = %d"
.text:00401057                 call    _printf
.text:0040105C                 add     esp, 8
.text:0040105F                 xor     eax, eax
.text:00401061                 pop     ebp
.text:00401062                 retn
.text:00401062 _main           endp
```

4.4.8.除数为有符号负非2的幂

release版

```
#include <stdio.h>
int main(int  argc, char* argv[]) {

    printf("a / -5 = %d", argc / -5);

    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 mov     eax, 99999999h  ; 魔改数0x999999999超过0x7fffffff，说明除数为负数
.text:00401048                 imul    [ebp+argc]
.text:0040104B                 sar     edx, 1
.text:0040104D                 mov     eax, edx
.text:0040104F                 shr     eax, 31
.text:00401052                 add     eax, edx
.text:00401054                 push    eax
.text:00401055                 push    offset _Format  ; "a / -5 = %d"
.text:0040105A                 call    _printf
.text:0040105F                 add     esp, 8
.text:00401062                 xor     eax, eax
.text:00401064                 pop     ebp
.text:00401065                 retn
.text:00401065 _main           endp
```

总结

$$|c| = \frac{2^n}{2^{32} - M}$$

- 魔改数0x999999999 > 0x7fffffff，说明除数为负数
- 除数 = 2^{32+1} / 2^{32} - 2576980377 = 5 , 再取负数为 -5

4.4.9.另一种除数为有符号负非2的幂

release版

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    printf("argc / -7 = %d", argc / -7); //变量除以常量，常量为负非2的幂
    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 mov     eax, 6DB6DB6Dh  ; M = 1840700269
.text:00401048                 imul    [ebp+argc]      ; edx.eax = argc*M
.text:0040104B                 sub     edx, [ebp+argc] ; edx = (argc*M >> 32) - argc
.text:0040104E                 sar     edx, 2          ; edx = edx = ((argc*M >> 32) - argc) >> 2
.text:00401051                 mov     eax, edx
.text:00401053                 shr     eax, 31         ; 取符号位
.text:00401056                 add     eax, edx
.text:00401058                 push    eax
.text:00401059                 push    offset _Format  ; "argc / -7 = %d"
.text:0040105E                 call    _printf
.text:00401063                 add     esp, 8
.text:00401066                 xor     eax, eax
.text:00401068                 pop     ebp
.text:00401069                 retn
.text:00401069 _main           endp
.text:00401069
```

总结

$$|c| = \frac{2^n}{2^{32} - M}$$

- 魔数取值小于等于7fffffffffffffh，而imul和sar之间有sub指令调整乘积，故可认定除数为负
- 除数 = 2^{32+2} / 2^{32} - 2576980377 = 17,179,869,184 / 2,454,267,027 = 6.9999

4.5.取模

release版

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    printf("%d", argc % 8); //变量模常量，常量为2的幂
    printf("%d", argc % 9); //变量模常量，常量为非2的幂
    return 0;
}
```

ida反汇编

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 push    esi
.text:00401044                 mov     esi, [ebp+argc] ; esi = argc
.text:00401047                 mov     eax, esi
.text:00401049                 and     eax, 80000007h  ; eax = argc & 7 (最高位1为了检查负数)，统计低位保留多少个1，即可得到k值
.text:0040104E                 jns     short loc_401055 ; if (argc >= 0) 则跳转
.text:00401050                 dec     eax
.text:00401051                 or      eax, 0FFFFFFF8h ; 统计低位保留多少个0，即可得到k值
.text:00401054                 inc     eax             ; if (argc < 0)  eax= ((argc & 7) -1 | -7) + 1
.text:00401055
.text:00401055 loc_401055:                             ; CODE XREF: _main+E↑j
.text:00401055                 push    eax
.text:00401056                 push    offset _Format  ; "%d"
.text:0040105B                 call    _printf
.text:00401060                 mov     eax, 38E38E39h
.text:00401065                 imul    esi
.text:00401067                 sar     edx, 1
.text:00401069                 mov     eax, edx
.text:0040106B                 shr     eax, 31
.text:0040106E                 add     eax, edx
.text:00401070                 lea     eax, [eax+eax*8]
.text:00401073                 sub     esi, eax
.text:00401075                 push    esi
.text:00401076                 push    offset _Format  ; "%d"
.text:0040107B                 call    _printf
.text:00401080                 add     esp, 10h
.text:00401083                 xor     eax, eax
.text:00401085                 pop     esi
.text:00401086                 pop     ebp
.text:00401087                 retn
.text:00401087 _main           endp
```

总结

- 第一种对2的k次方取余：and eax,80000007h，去掉高位保留低位，统计低位一个保留了多少个1（7的二进制位0111，保留了3个1），即可得到k的值为3，然后得到结果：$2^3 = 8$
- 第二种对非2的k次方取余：[eax+eax*8] = eax * 9 ,即可得到结果9

4.6.条件跳转指令表

指令助记符	检查标记位	说明
JZ	ZF == 1	等于 0 则跳转
JE	ZF == 1	相等则跳转
JNZ	ZF == 0	不等于 0 则跳转
JNE	ZF == 0	不相等则跳转
JS	SF == 1	符号为负则跳转
JNS	SF == 0	符号为正则跳转
JP/JPE	PF == 1	“1”的个数为偶数则跳转
JNP/JPO	PF == 0	“1”的个数为奇数则跳转
JO	OF == 1	溢出则跳转
JNO	OF == 0	无溢出则跳转
JC	CF == 1	进位则跳转
JB	CF == 1	小于则跳转
JNAE	CF == 1	不大于等于则跳转
JNB	CF == 0	不小于则跳转
JAE	CF == 0	大于等于则跳转
JBE	CF == 1 或 ZF == 1	小于等于则跳转
JNA	CF == 1 或 ZF == 1	不大于则跳转
JNBE	CF == 0 或 ZF == 0	不小于等于则跳转
JA	CF == 0 或 ZF == 0	大于则跳转
JL	SF != OF	小于则跳转
JNGE	SF != OF	不大于等于则跳转
JNL	SF == OF	不小于则跳转
JGE	SF == OF	大于等于则跳转
JLE	ZF != OF 或 ZF == 1	小于等于则跳转
JNG	ZF != OF 或 ZF == 1	不大于则跳转
JNLE	SF ==OF 且 ZF == 0	不小于等于则跳转
JG	SF ==OF 且 ZF == 0	大于则跳转

4.7.条件表达式

第一种，相差为1

release版

```
#include <stdio.h>


int main(int argc, char* argv[])
{
    printf("%d\r\n",argc == 5 ? 5:6);

    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 xor     eax, eax
.text:00401045                 cmp     [ebp+argc], 5
.text:00401049                 setnz   al              ; if (argc=5) al = 0 , else al = 1
.text:0040104C                 add     eax, 5          ; if (argc = 5), eax = 5;  else eax = 5 + 1 = 6
.text:0040104F                 push    eax
.text:00401050                 push    offset _Format  ; "%d\n"
.text:00401055                 call    _printf
.text:0040105A                 add     esp, 8
.text:0040105D                 xor     eax, eax
.text:0040105F                 pop     ebp
.text:00401060                 retn
.text:00401060 _main           endp
```

第二种,相差大于1

release

```
#include <stdio.h>


int main(int argc, char* argv[])
{
    printf("%d\r\n",argc == 5 ? 4:10);

    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 cmp     [ebp+argc], 5
.text:00401047                 mov     ecx, 4
.text:0040104C                 mov     eax, 10
.text:00401051                 cmovz   eax, ecx        ; if (ZF=1), eax=ecx=4, else eax=10
.text:00401054                 push    eax
.text:00401055                 push    offset _Format  ; "%d\n"
.text:0040105A                 call    _printf
.text:0040105F                 add     esp, 8
.text:00401062                 xor     eax, eax
.text:00401064                 pop     ebp
.text:00401065                 retn
.text:00401065 _main           endp
```

第三种变量表达式

release

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    int n1, n2;
    scanf_s("%d %d", &n1, &n2);
    printf("%d\n", argc ? n1 : n2);
    return 0;
}
```

ida

```
.text:00401090 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401090 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401090
.text:00401090 n1              = dword ptr -0Ch
.text:00401090 n2              = dword ptr -8
.text:00401090 var_4           = dword ptr -4
.text:00401090 argc            = dword ptr  8
.text:00401090 argv            = dword ptr  0Ch
.text:00401090 envp            = dword ptr  10h
.text:00401090
.text:00401090                 push    ebp
.text:00401091                 mov     ebp, esp
.text:00401093                 sub     esp, 0Ch
.text:00401096                 mov     eax, ___security_cookie
.text:0040109B                 xor     eax, ebp
.text:0040109D                 mov     [ebp+var_4], eax
.text:004010A0                 lea     eax, [ebp+n2]
.text:004010A3                 push    eax
.text:004010A4                 lea     eax, [ebp+n1]
.text:004010A7                 push    eax
.text:004010A8                 push    offset _Format  ; "%d %d"
.text:004010AD                 call    _scanf_s
.text:004010B2                 mov     eax, [ebp+n2]
.text:004010B5                 cmp     [ebp+argc], 0
.text:004010B9                 cmovnz  eax, [ebp+n1]   ; if (argc!= 0)  eax = n1, else eax = n2
.text:004010BD                 push    eax
.text:004010BE                 push    offset aD       ; "%d\n"
.text:004010C3                 call    _printf
.text:004010C8                 mov     ecx, [ebp+var_4]
.text:004010CB                 add     esp, 14h
.text:004010CE                 xor     ecx, ebp        ; StackCookie
.text:004010D0                 xor     eax, eax
.text:004010D2                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:004010D7                 mov     esp, ebp
.text:004010D9                 pop     ebp
.text:004010DA                 retn
.text:004010DA _main           endp
```

第四种表达式无优化使用分支

release

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    int n1, n2;
    scanf_s("%d %d", &n1, &n2);
    printf("%d\n", argc ? n1 : n2 + 3);
    return 0;
}
```

ida

```
.text:00401090 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401090 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401090
.text:00401090 n2              = dword ptr -0Ch
.text:00401090 n1              = dword ptr -8
.text:00401090 var_4           = dword ptr -4
.text:00401090 argc            = dword ptr  8
.text:00401090 argv            = dword ptr  0Ch
.text:00401090 envp            = dword ptr  10h
.text:00401090
.text:00401090                 push    ebp
.text:00401091                 mov     ebp, esp
.text:00401093                 sub     esp, 0Ch
.text:00401096                 mov     eax, ___security_cookie
.text:0040109B                 xor     eax, ebp
.text:0040109D                 mov     [ebp+var_4], eax
.text:004010A0                 lea     eax, [ebp+n2]
.text:004010A3                 push    eax
.text:004010A4                 lea     eax, [ebp+n1]
.text:004010A7                 push    eax
.text:004010A8                 push    offset _Format  ; "%d %d"
.text:004010AD                 call    _scanf_s
.text:004010B2                 add     esp, 12
.text:004010B5                 cmp     [ebp+argc], 0
.text:004010B9                 jz      short loc_4010C0 ; 使用流程语句进行比较和判断
.text:004010BB                 mov     eax, [ebp+n1]
.text:004010BE                 jmp     short loc_4010C6
.text:004010C0 ; ---------------------------------------------------------------------------
.text:004010C0
.text:004010C0 loc_4010C0:                             ; CODE XREF: _main+29↑j
.text:004010C0                 mov     eax, [ebp+n2]
.text:004010C3                 add     eax, 3
.text:004010C6
.text:004010C6 loc_4010C6:                             ; CODE XREF: _main+2E↑j
.text:004010C6                 push    eax
.text:004010C7                 push    offset aD       ; "%d\n"
.text:004010CC                 call    _printf
.text:004010D1                 mov     ecx, [ebp+var_4]
.text:004010D4                 add     esp, 8
.text:004010D7                 xor     ecx, ebp        ; StackCookie
.text:004010D9                 xor     eax, eax
.text:004010DB                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:004010E0                 mov     esp, ebp
.text:004010E2                 pop     ebp
.text:004010E3                 retn
.text:004010E3 _main           endp
```

五、流程控制语句的识别

5.1.if

release

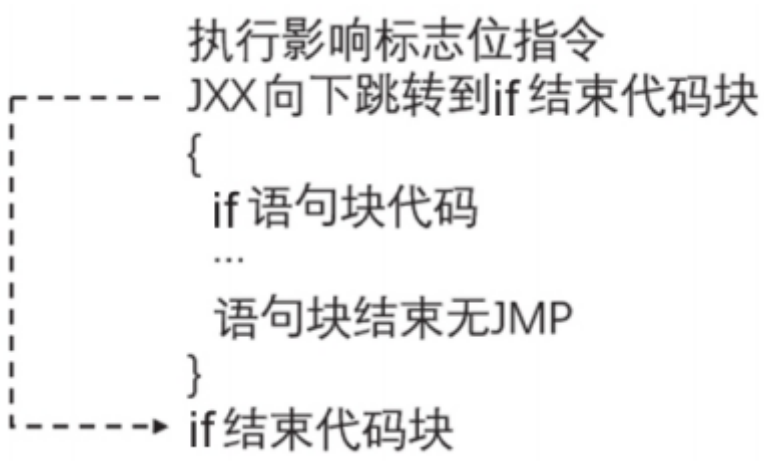
```
#include <stdio.h>

int main(int argc, char* argv[]) {
    if (argc == 0) {
        printf("argc == 0");
    }
    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 cmp     [ebp+argc], 0
.text:00401047                 jnz     short loc_401056 ; if语句转换的条件跳转指令与if语句的判断结果是相反的
.text:00401049                 push    offset _Format  ; "argc == 0"
.text:0040104E                 call    _printf
.text:00401053                 add     esp, 4          ; if语句块代码
.text:00401056
.text:00401056 loc_401056:                             ; CODE XREF: _main+7↑j
.text:00401056                 xor     eax, eax        ; if结束块代码
.text:00401058                 pop     ebp
.text:00401059                 retn
.text:00401059 _main           endp
```

总结



5.2.if else

release

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    if (argc > 0) {
        printf("argc > 0");
    }
    else if (argc == 0) {
        printf("argc == 0");
    }
    else {
        printf("argc <= 0");
    }
    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 mov     ecx, [ebp+argc]
.text:00401046                 test    ecx, ecx
.text:00401048                 jle     short loc_40105C ; 如果argc<=0，则跳转到if结束代码块
.text:0040104A                 mov     eax, offset _Format ; "argc > 0"
.text:0040104F                 push    eax             ; _Format
.text:00401050                 call    _printf
.text:00401055                 add     esp, 4
.text:00401058                 xor     eax, eax
.text:0040105A                 pop     ebp
.text:0040105B                 retn
.text:0040105C ; ---------------------------------------------------------------------------
.text:0040105C
.text:0040105C loc_40105C:                             ; CODE XREF: _main+8↑j
.text:0040105C                 test    ecx, ecx        ; if结束代码块
.text:0040105E                 mov     edx, offset aArgc0_0 ; "argc <= 0"
.text:00401063                 mov     eax, offset aArgc0_1 ; "argc == 0"
.text:00401068                 cmovnz  eax, edx        ; 无分支优化
.text:0040106B                 push    eax             ; _Format
.text:0040106C                 call    _printf
.text:00401071                 add     esp, 4
.text:00401074                 xor     eax, eax
.text:00401076                 pop     ebp
.text:00401077                 retn
.text:00401077 _main           endp
```

5.3.switch

5.3.1.分支少于4个

release

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    int n = 1;
    scanf_s("%d", &n);
    switch (n) {
    case 1:
        printf("n == 1");
        break;
    case 3:
        printf("n == 3");
        break;
    case 100:
        printf("n == 100");
        break;
    }
    return 0;
}
```

ida

```
.text:00401090 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401090 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401090
.text:00401090 n               = dword ptr -8
.text:00401090 var_4           = dword ptr -4
.text:00401090 argc            = dword ptr  8
.text:00401090 argv            = dword ptr  0Ch
.text:00401090 envp            = dword ptr  10h
.text:00401090
.text:00401090                 push    ebp
.text:00401091                 mov     ebp, esp
.text:00401093                 sub     esp, 8
.text:00401096                 mov     eax, ___security_cookie
.text:0040109B                 xor     eax, ebp
.text:0040109D                 mov     [ebp+var_4], eax
.text:004010A0                 lea     eax, [ebp+n]
.text:004010A3                 mov     [ebp+n], 1
.text:004010AA                 push    eax
.text:004010AB                 push    offset _Format  ; "%d"
.text:004010B0                 call    _scanf_s
.text:004010B5                 mov     eax, [ebp+n]
.text:004010B8                 add     esp, 8
.text:004010BB                 sub     eax, 1          ; n = 1
.text:004010BE                 jz      short loc_4010D8
.text:004010C0                 sub     eax, 2          ; n = 3
.text:004010C3                 jz      short loc_4010D1
.text:004010C5                 sub     eax, 97         ; n = 100
.text:004010C8                 jnz     short loc_4010E5
.text:004010CA                 push    offset aN100    ; "n == 100"
.text:004010CF                 jmp     short loc_4010DD
.text:004010D1 ; ---------------------------------------------------------------------------
.text:004010D1 ; ---------------------------------------------------------------------------
.text:004010D1
.text:004010D1 loc_4010D1:                             ; CODE XREF: _main+33↑j
.text:004010D1                 push    offset aN3      ; "n == 3"
.text:004010D6                 jmp     short loc_4010DD
.text:004010D8 ; ---------------------------------------------------------------------------
.text:004010D8
.text:004010D8 loc_4010D8:                             ; CODE XREF: _main+2E↑j
.text:004010D8                 push    offset aN1      ; "n == 1"
.text:004010DD
.text:004010DD loc_4010DD:                             ; CODE XREF: _main+3F↑j
.text:004010DD                                         ; _main+46↑j
.text:004010DD                 call    _printf
.text:004010E2                 add     esp, 4
.text:004010E5
.text:004010E5 loc_4010E5:                             ; CODE XREF: _main+38↑j
.text:004010E5                 mov     ecx, [ebp+var_4]
.text:004010E8                 xor     eax, eax
.text:004010EA                 xor     ecx, ebp        ; StackCookie
.text:004010EC                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:004010F1                 mov     esp, ebp
.text:004010F3                 pop     ebp
.text:004010F4                 retn
.text:004010F4 _main           endp
```

5.3.2.分支大于4个且值连续

会对case语句块制作地址表，以减少比较跳转次数

release

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int n = 1;
    scanf_s("%d", &n);
    switch (n) {
    case 1:
        printf("n == 1");
        break;
    case 2:
```

```
        printf("n == 2");
        break;
    case 3:
        printf("n == 3");
        break;
    case 5:
        printf("n == 5");
        break;
    case 6:
        printf("n == 6");
        break;
    case 7:
        printf("n == 7");
        break;
    }
    return 0;
}
```

ida

```
.text:00401090 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401090 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401090
.text:00401090 n               = dword ptr -8
.text:00401090 var_4           = dword ptr -4
.text:00401090 argc            = dword ptr  8
.text:00401090 argv            = dword ptr  0Ch
.text:00401090 envp            = dword ptr  10h
.text:00401090
.text:00401090                 push    ebp
.text:00401091                 mov     ebp, esp
.text:00401093                 sub     esp, 8
.text:00401096                 mov     eax, ___security_cookie
.text:0040109B                 xor     eax, ebp
.text:0040109D                 mov     [ebp+var_4], eax
.text:004010A0                 lea     eax, [ebp+n]
.text:004010A3                 mov     [ebp+n], 1
.text:004010AA                 push    eax
.text:004010AB                 push    offset _Format  ; "%d"
.text:004010B0                 call    _scanf_s
.text:004010B5                 mov     eax, [ebp+n]
.text:004010B8                 add     esp, 8
.text:004010BB                 dec     eax             ; switch 7 cases
.text:004010BC                 cmp     eax, 6
.text:004010BF                 ja      short def_4010C1 ; if n >7 则跳转到switch结束代码块
.text:004010C1                 jmp     ds:jpt_4010C1[eax*4] ; 查表
.text:004010C8 ; ---------------------------------------------------------------------------
.text:004010C8
.text:004010C8 CASE1:                                  ; CODE XREF: _main+31↑j
.text:004010C8                                         ; DATA XREF: _main:jpt_4010C1↓o
.text:004010C8                 push    offset aN1      ; jumptable 004010C1 case 1
.text:004010CD                 jmp     short loc_4010F0
.text:004010CF ; ---------------------------------------------------------------------------
.text:004010CF
.text:004010CF CASE2:                                  ; CODE XREF: _main+31↑j
.text:004010CF                                         ; DATA XREF: _main:jpt_4010C1↓o
.text:004010CF                 push    offset aN2      ; jumptable 004010C1 case 2
.text:004010D4                 jmp     short loc_4010F0
.text:004010D6 ; ---------------------------------------------------------------------------
.text:004010D6
.text:004010D6 CASE3:                                  ; CODE XREF: _main+31↑j
.text:004010D6                                         ; DATA XREF: _main:jpt_4010C1↓o
.text:004010D6                 push    offset aN3      ; jumptable 004010C1 case 3
.text:004010DB                 jmp     short loc_4010F0
.text:004010DD ; ---------------------------------------------------------------------------
```

```
.text:004010DD CASE5:                                  ; CODE XREF: _main+31↑j
.text:004010DD                                         ; DATA XREF: _main:jpt_4010C1↓o
.text:004010DD                 push    offset aN5      ; jumptable 004010C1 case 5
.text:004010E2                 jmp     short loc_4010F0
.text:004010E4 ; ---------------------------------------------------------------------------
.text:004010E4
.text:004010E4 CASE6:                                  ; CODE XREF: _main+31↑j
.text:004010E4                                         ; DATA XREF: _main:jpt_4010C1↓o
.text:004010E4                 push    offset aN6      ; jumptable 004010C1 case 6
.text:004010E9                 jmp     short loc_4010F0
.text:004010EB ; ---------------------------------------------------------------------------
.text:004010EB
.text:004010EB CASE7:                                  ; CODE XREF: _main+31↑j
.text:004010EB                                         ; DATA XREF: _main:jpt_4010C1↓o
.text:004010EB                 push    offset aN7      ; jumptable 004010C1 case 7
.text:004010F0
.text:004010F0 loc_4010F0:                             ; CODE XREF: _main+3D↑j
.text:004010F0                                         ; _main+44↑j ...
.text:004010F0                 call    _printf
.text:004010F5                 add     esp, 4
.text:004010F8
.text:004010F8 def_4010C1:                             ; CODE XREF: _main+2F↑j
.text:004010F8                                         ; _main+31↑j
.text:004010F8                                         ; DATA XREF: ...
.text:004010F8                 mov     ecx, [ebp+var_4] ; jumptable 004010C1 default case, case 4
.text:004010FB                 xor     eax, eax
.text:004010FD                 xor     ecx, ebp        ; StackCookie
.text:004010FF                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:00401104                 mov     esp, ebp
.text:00401106                 pop     ebp
.text:00401107                 retn
.text:00401107 ; ---------------------------------------------------------------------------
.text:00401108 jpt_4010C1      dd offset CASE1         ; DATA XREF: _main+31↑r
.text:00401108                 dd offset CASE2         ; jump table for switch statement
.text:00401108                 dd offset CASE3
.text:00401108                 dd offset def_4010C1
.text:00401108                 dd offset CASE5
.text:00401108                 dd offset CASE6
.text:00401108                 dd offset CASE7
.text:00401108 _main           endp
```

5.3.3.分支大于4个，值不连续，且最大case值和case值的差小于256

有两张表

- case语句块地址表：每一项保存一个case语句块的首地址，有几个case就有几项，default也在里面
- case语句块索引表：保存地址表的编号，索引表的大小等于最大case值和最小case值的差

release版

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int n = 1;
    scanf_s("%d", &n);
    switch (n) {
    case 1:
        printf("n == 1");
        break;
    case 2:
        printf("n == 2");
        break;
    case 3:
        printf("n == 3");
        break;
    case 5:
        printf("n == 5");
```

```
        break;
    case 6:
        printf("n == 6");
        break;
    case 255:
        printf("n == 255");
        break;
    }
    return 0;
}
```

ida

```
.text:00401090 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401090 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401090
.text:00401090 n               = dword ptr -8
.text:00401090 var_4           = dword ptr -4
.text:00401090 argc            = dword ptr  8
.text:00401090 argv            = dword ptr  0Ch
.text:00401090 envp            = dword ptr  10h
.text:00401090
.text:00401090                 push    ebp
.text:00401091                 mov     ebp, esp
.text:00401093                 sub     esp, 8
.text:00401096                 mov     eax, ___security_cookie
.text:0040109B                 xor     eax, ebp
.text:0040109D                 mov     [ebp+var_4], eax
.text:004010A0                 lea     eax, [ebp+n]
.text:004010A3                 mov     [ebp+n], 1
.text:004010AA                 push    eax
.text:004010AB                 push    offset _Format  ; "%d"
.text:004010B0                 call    _scanf_s
.text:004010B5                 mov     eax, [ebp+n]
.text:004010B8                 add     esp, 8
.text:004010BB                 dec     eax             ; switch 255 cases
.text:004010BC                 cmp     eax, 254
.text:004010C1                 ja      short CASE_END  ; jumptable 004010CA default case, cases 4,7-254
.text:004010C3                 movzx   eax, ds:byte_401130[eax] ; case索引表
.text:004010CA                 jmp     ds:jpt_4010CA[eax*4] ; switch jump
.text:004010D1 ; ---------------------------------------------------------------------------
.text:004010D1
.text:004010D1 CASE1:                                  ; CODE XREF: _main+3A↑j
.text:004010D1                                         ; DATA XREF: _main:jpt_4010CA↓o
.text:004010D1                 push    offset aN1      ; jumptable 004010CA case 1
.text:004010D6                 jmp     short loc_4010F9
.text:004010D8 ; ---------------------------------------------------------------------------
.text:004010D8
.text:004010D8 CASE2:                                  ; CODE XREF: _main+3A↑j
.text:004010D8                                         ; DATA XREF: _main:jpt_4010CA↓o
.text:004010D8                 push    offset aN2      ; jumptable 004010CA case 2
.text:004010DD                 jmp     short loc_4010F9
.text:004010DF ; ---------------------------------------------------------------------------
```

```
.text:004010DF
.text:004010DF CASE3:                                  ; CODE XREF: _main+3A↑j
.text:004010DF                                         ; DATA XREF: _main:jpt_4010CA↓o
.text:004010DF                 push    offset aN3      ; jumptable 004010CA case 3
.text:004010E4                 jmp     short loc_4010F9
.text:004010E6 ; ---------------------------------------------------------------------------
.text:004010E6
.text:004010E6 CASE5:                                  ; CODE XREF: _main+3A↑j
.text:004010E6                                         ; DATA XREF: _main:jpt_4010CA↓o
.text:004010E6                 push    offset aN5      ; jumptable 004010CA case 5
.text:004010EB                 jmp     short loc_4010F9
.text:004010ED ; ---------------------------------------------------------------------------
.text:004010ED
.text:004010ED CASE6:                                  ; CODE XREF: _main+3A↑j
.text:004010ED                                         ; DATA XREF: _main:jpt_4010CA↓o
.text:004010ED                 push    offset aN6      ; jumptable 004010CA case 6
.text:004010F2                 jmp     short loc_4010F9
.text:004010F4 ; ---------------------------------------------------------------------------
.text:004010F4
.text:004010F4 CASE255:                                ; CODE XREF: _main+3A↑j
.text:004010F4                                         ; DATA XREF: _main:jpt_4010CA↓o
.text:004010F4                 push    offset aN255    ; jumptable 004010CA case 255
.text:004010F9
.text:004010F9 loc_4010F9:                             ; CODE XREF: _main+46↑j
.text:004010F9                                         ; _main+4D↑j ...
.text:004010F9                 call    _printf
.text:004010FE                 add     esp, 4
.text:00401101
.text:00401101 CASE_END:                               ; CODE XREF: _main+31↑j
.text:00401101                                         ; _main+3A↑j
.text:00401101                                         ; DATA XREF: ...
.text:00401101                 mov     ecx, [ebp+var_4] ; jumptable 004010CA default case, cases 4,7-254
.text:00401104                 xor     eax, eax
.text:00401106                 xor     ecx, ebp        ; StackCookie
.text:00401108                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:0040110D                 mov     esp, ebp
.text:0040110F                 pop     ebp
.text:00401110                 retn
.text:00401110 ; ---------------------------------------------------------------------------
```

```
.text:00401111                 align 4                 ; case地址表
.text:00401114 jpt_4010CA      dd offset CASE1         ; DATA XREF: _main+3A↑r
.text:00401114                 dd offset CASE2         ; jump table for switch statement
.text:00401114                 dd offset CASE3
.text:00401114                 dd offset CASE5
.text:00401114                 dd offset CASE6
.text:00401114                 dd offset CASE255
.text:00401114                 dd offset CASE_END
.text:00401130 byte_401130     db 0, 1, 2, 6, 3, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                                         ; DATA XREF: _main+33↑r
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6 ; case索引表
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6
.text:00401130                 db 6, 6, 5
.text:00401130 _main           endp
```

5.3.4.分支大于4个，值不连续，且最大case值和case值的差大于256

将每个case值作为一个节点，找到这些节点的中间值作为跟节点，形成一颗平衡二叉树，以每个节点作为判定值，大于和小于关系分别对应左子树和右子树。

release版

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int n = 0;
    scanf_s("%d", &n);
    switch (n) {
    case 2:
```

```
        printf("n == 2\n");
        break;
    case 3:
        printf("n == 3\n");
        break;
    case 8:
        printf("n == 8\n");
        break;
    case 10:
        printf("n == 10\n");
        break;
    case 35:
        printf("n == 35\n");
        break;
    case 37:
        printf("n == 37\n");
        break;
    case 666:
        printf("n == 666\n");
        break;
    default:
        printf("default\n");
        break;
    }
    return 0;
}
```

ida

```
.text:00401090 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401090 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401090
.text:00401090 n               = dword ptr -8
.text:00401090 var_4           = dword ptr -4
.text:00401090 argc            = dword ptr  8
.text:00401090 argv            = dword ptr  0Ch
.text:00401090 envp            = dword ptr  10h
.text:00401090
.text:00401090                 push    ebp
.text:00401091                 mov     ebp, esp
.text:00401093                 sub     esp, 8
.text:00401096                 mov     eax, ___security_cookie
.text:0040109B                 xor     eax, ebp
.text:0040109D                 mov     [ebp+var_4], eax
.text:004010A0                 lea     eax, [ebp+n]
.text:004010A3                 mov     [ebp+n], 0
.text:004010AA                 push    eax
.text:004010AB                 push    offset _Format  ; "%d"
.text:004010B0                 call    _scanf_s
.text:004010B5                 mov     eax, [ebp+n]
.text:004010B8                 add     esp, 8
.text:004010BB                 cmp     eax, 10
.text:004010BE                 jg      short loc_4010ED ; n > 10
.text:004010C0                 jz      short loc_4010E6 ; n = 10
.text:004010C2                 sub     eax, 2
.text:004010C5                 jz      short loc_4010DF ; n = 2
.text:004010C7                 sub     eax, 1           ; n = 3
.text:004010CA                 jz      short loc_4010D8
.text:004010CC                 sub     eax, 5
.text:004010CF                 jnz     short loc_4010FE
.text:004010D1                 mov     eax, offset aN8 ; "n == 8\n"
.text:004010D6                 jmp     short loc_401118
.text:004010D8 ; ---------------------------------------------------------------------------
.text:004010D8
.text:004010D8 loc_4010D8:                             ; CODE XREF: _main+3A↑j
.text:004010D8                 mov     eax, offset aN3 ; "n == 3\n"
.text:004010DD                 jmp     short loc_401118
.text:004010DF ; ---------------------------------------------------------------------------
.text:004010DF
.text:004010DF loc_4010DF:                             ; CODE XREF: _main+35↑j
.text:004010DF                 mov     eax, offset aN2 ; "n == 2\n"
.text:004010E4                 jmp     short loc_401118
.text:004010E6 ; ---------------------------------------------------------------------------
.text:004010E6
```

```
.text:004010E6 loc_4010E6:                             ; CODE XREF: _main+30↑j
.text:004010E6                 mov     eax, offset aN10 ; "n == 10\n"
.text:004010EB                 jmp     short loc_401118
.text:004010ED ; ---------------------------------------------------------------------------
.text:004010ED
.text:004010ED loc_4010ED:                             ; CODE XREF: _main+2E↑j
.text:004010ED                 sub     eax, 35         ; n = 35
.text:004010F0                 jz      short loc_401113
.text:004010F2                 sub     eax, 2          ; n = 37
.text:004010F5                 jz      short loc_40110C
.text:004010F7                 sub     eax, 629        ; n = 666
.text:004010FC                 jz      short loc_401105
.text:004010FE
.text:004010FE loc_4010FE:                             ; CODE XREF: _main+3F↑j
.text:004010FE                 mov     eax, offset aDefault ; "default\n"
.text:00401103                 jmp     short loc_401118
.text:00401105 ; ---------------------------------------------------------------------------
.text:00401105
.text:00401105 loc_401105:                             ; CODE XREF: _main+6C↑j
.text:00401105                 mov     eax, offset aN666 ; "n == 666\n"
.text:0040110A                 jmp     short loc_401118
.text:0040110C ; ---------------------------------------------------------------------------
.text:0040110C
.text:0040110C loc_40110C:                             ; CODE XREF: _main+65↑j
.text:0040110C                 mov     eax, offset aN37 ; "n == 37\n"
.text:00401111                 jmp     short loc_401118
.text:00401113 ; ---------------------------------------------------------------------------
.text:00401113
.text:00401113 loc_401113:                             ; CODE XREF: _main+60↑j
.text:00401113                 mov     eax, offset aN35 ; "n == 35\n"
.text:00401118
.text:00401118 loc_401118:                             ; CODE XREF: _main+46↑j
.text:00401118                                         ; _main+4D↑j ...
.text:00401118                 push    eax             ; _Format
.text:00401119                 call    _printf
.text:0040111E                 mov     ecx, [ebp+var_4]
.text:00401121                 add     esp, 4
.text:00401124                 xor     ecx, ebp        ; StackCookie
.text:00401126                 xor     eax, eax
.text:00401128                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:0040112D                 mov     esp, ebp
.text:0040112F                 pop     ebp
.text:00401130                 retn
.text:00401130 _main           endp
```

5.4.do while

release

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    int sum = 0;
    int i = 0;
    do {
        sum += i;
        i++;
    } while (i <= argc);
    return sum;
}
```

ida

```
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401000
.text:00401000 argc            = dword ptr  8
.text:00401000 argv            = dword ptr  0Ch
.text:00401000 envp            = dword ptr  10h
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 mov     edx, [ebp+argc]
.text:00401006                 xor     eax, eax
.text:00401008                 xor     ecx, ecx
.text:0040100A                 nop     word ptr [eax+eax+00h]
.text:00401010
.text:00401010 loc_401010:                             ; CODE XREF: _main+15↓j
.text:00401010                 add     eax, ecx        ; sum += i
.text:00401012                 inc     ecx             ; i++
.text:00401013                 cmp     ecx, edx        ; i <= argc
.text:00401015                 jle     short loc_401010 ; sum += i
.text:00401017                 pop     ebp
.text:00401018                 retn
.text:00401018 _main           endp
```

5.5.while

release版

```
#include <stdio.h>


int main(int argc, char* argv[])
{
    int sum = 0;
    int i = 0;
    while (i <= 100)
    {
        sum = sum + i;
        i++;
    }

    printf("%d\r\n", sum);

    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040
.text:00401040                 xor     ecx, ecx
.text:00401042                 xor     edx, edx
.text:00401044                 xor     eax, eax
.text:00401046
.text:00401046 loc_401046:                             ; CODE XREF: _main+11↓j
.text:00401046                 inc     edx
.text:00401047                 add     ecx, eax
.text:00401049                 add     edx, eax
.text:0040104B                 add     eax, 2          ; 步长为2，减少循环次数
.text:0040104E                 cmp     eax, 99
.text:00401051                 jle     short loc_401046
.text:00401053                 push    esi
.text:00401054                 xor     esi, esi
.text:00401056                 cmp     eax, 100
.text:00401059                 cmovg   eax, esi
.text:0040105C                 add     eax, edx
.text:0040105E                 add     eax, ecx
.text:00401060                 push    eax
.text:00401061                 push    offset _Format  ; "%d\r\n"
.text:00401066                 call    _printf
.text:0040106B                 add     esp, 8
.text:0040106E                 xor     eax, eax
.text:00401070                 pop     esi
.text:00401071                 retn
.text:00401071 _main           endp
.text:00401071
```

5.6.for

main.cpp

```cpp
#include <stdio.h>


int main(int argc, char* argv[])
{
    int sum = 0;

    //内部会优化，把步长改为4，减少循环次数
    for (int n = 1; n <= 100; n++)
    {
        sum = sum + n;
    }

    printf("%d\r\n", sum);

    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040
.text:00401040                 push    esi
.text:00401041                 push    edi
.text:00401042                 xor     ecx, ecx
.text:00401044                 xor     edx, edx
.text:00401046                 xor     esi, esi
.text:00401048                 mov     eax, 1
.text:0040104D                 xor     edi, edi
.text:0040104F                 nop
.text:00401050
.text:00401050 loc_401050:                             ; CODE XREF: _main+25↓j
.text:00401050                 inc     edi
.text:00401051                 add     esi, 2
.text:00401054                 add     edx, 3
.text:00401057                 add     ecx, eax
.text:00401059                 add     edi, eax
.text:0040105B                 add     esi, eax
.text:0040105D                 add     edx, eax
.text:0040105F                 add     eax, 4          ; 步长为4，减少循环次数
.text:00401062                 cmp     eax, 64h ; 'd'
.text:00401065                 jle     short loc_401050
.text:00401067                 lea     eax, [edx+esi]
.text:0040106A                 add     eax, edi
.text:0040106C                 add     ecx, eax
.text:0040106E                 push    ecx
.text:0040106F                 push    offset _Format  ; "%d\r\n"
.text:00401074                 call    _printf
.text:00401079                 add     esp, 8
.text:0040107C                 xor     eax, eax
.text:0040107E                 pop     edi
.text:0040107F                 pop     esi
.text:00401080                 retn
.text:00401080 _main           endp
```

六、函数的工作原理

6.1.各种调用方式的考察

调用约定

- _cdecl：默认的调用约定，外平栈，按从右至左的顺序压参数入栈
- _stdcall：内平栈，按从右至左的顺序压参数入栈
- _fastcall：：前两个参数用ecx和edx传参，其余参数通过栈传参方式，按从右至左的顺序压参数入栈

6.2.函数的参数

release

```
#include <stdio.h>

void addNumber(int n1) {
    n1 += 1;
    printf("%d\n", n1);
}
int main(int argc, char* argv[]) {
    int n = 0;
    scanf_s("%d", &n); // 防止变量被常量扩散优化
    addNumber(n);
    return 0;
}
```

ida

```
.text:00401090 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401090 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401090
.text:00401090 n               = dword ptr -8
.text:00401090 var_4           = dword ptr -4
.text:00401090 argc            = dword ptr  8
.text:00401090 argv            = dword ptr  0Ch
.text:00401090 envp            = dword ptr  10h
.text:00401090
.text:00401090                 push    ebp
.text:00401091                 mov     ebp, esp
.text:00401093                 sub     esp, 8
.text:00401096                 mov     eax, ___security_cookie
.text:0040109B                 xor     eax, ebp
.text:0040109D                 mov     [ebp+var_4], eax
.text:004010A0                 lea     eax, [ebp+n]    ; 取出局部变量的地址存入eax
.text:004010A3                 mov     [ebp+n], 0      ; 赋值为0
.text:004010AA                 push    eax             ; 压入eax作为参数，eax保存局部变量地址
.text:004010AB                 push    offset _Format  ; "%d"
.text:004010B0                 call    _scanf_s
.text:004010B5                 mov     eax, [ebp+n]    ; 取出数据到eax中
.text:004010B8                 inc     eax             ; eax = eax + 1
.text:004010B9                 push    eax
.text:004010BA                 push    offset aD       ; "%d\n"
.text:004010BF                 call    _printf
.text:004010C4                 mov     ecx, [ebp+var_4]
.text:004010C7                 add     esp, 10h
.text:004010CA                 xor     ecx, ebp        ; StackCookie
.text:004010CC                 xor     eax, eax
.text:004010CE                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:004010D3                 mov     esp, ebp
.text:004010D5                 pop     ebp
.text:004010D6                 retn
.text:004010D6 _main           endp
```

七、变量在内存中的位置和访问方式

变量	作用范围	生存周期
局部变量	所定义的函数或者所定义的复合语句	所定义的函数或者所定义的复合语句
全局变量	整个程序	整个程序运行期间
静态局部变量	所定义的函数	整个程序运行期间
静态全局变量	所定义的文件	整个程序运行期间

变量的作用域

- 全局变量：属于进程作用域，整个进程都能够访问到
- 静态变量：属于文件作用域，在当前源码文件内可以访问到
- 局部变量：属于函数作用域，在函数内可以访问到

7.1.全局变量和局部变量的区别

全局变量和局部变量的区别

- 全局变量：可以在程序中的任何文职使用
- 局部变量：局限于函数作用域内，若超出作用域，则由栈平衡操作释放局方局部变量的空间
- 局部变量：通过申请栈空间存放，利用栈指针ebp或esp间接访问，其地址是一个未知可变值
- 全局变量：与常量类似，通过立即数访问

7.2.局部静态变量的工作方式

局部静态变量

- 存放在静态存储区
- 作用域：所定义的函数
- 生命周期：持续到程序结束
- 只初始化一次

release版

```
#include <stdio.h>

void showStatic(int n) {
    static int g_static = n;     //定义局部静态变量，赋值为参数
    printf("%d\n", g_static);    //显示静态变量
}
int main(int argc, char* argv[]) {
    for (int i = 0; i < 5; i++) {
        showStatic(i);       //循环调用显示局部静态变量的函数，每次传入不同值
    }
    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040
.text:00401040                 mov     eax, large fs:2Ch
.text:00401046                 push    esi
.text:00401047                 push    edi
.text:00401048                 xor     esi, esi
.text:0040104A                 mov     edi, [eax]
.text:0040104C                 nop     dword ptr [eax+00h]
.text:00401050
.text:00401050 loc_401050:                             ; CODE XREF: _main+5D↓j
.text:00401050                 mov     eax, pOnce      ; showStatic函数
.text:00401055                 cmp     eax, [edi+4]
.text:0040105B                 jle     short loc_401086
.text:0040105D                 push    offset pOnce    ; pOnce
.text:00401062                 call    __Init_thread_header
.text:00401067                 add     esp, 4
.text:0040106A                 cmp     pOnce, 0FFFFFFFFh ; 检查局部静态变量是否初始化的标志
.text:00401071                 jnz     short loc_401086 ; 如果不为0FFFFFFFFh，表示局部静态变量已初始化，跳转到输出
.text:00401073                 push    offset pOnce    ; pOnce
.text:00401078                 mov     dword_4033B8, esi ; 初始化局部静态变量
.text:0040107E                 call    __Init_thread_footer ; 调用函数多线程同步函数设置初始化标志
.text:00401083                 add     esp, 4
.text:00401086
.text:00401086 loc_401086:                             ; CODE XREF: _main+1B↑j
.text:00401086                                         ; _main+31↑j
.text:00401086                 push    dword_4033B8
.text:0040108C                 push    offset _Format  ; "%d\n"
.text:00401091                 call    _printf
.text:00401096                 inc     esi
.text:00401097                 add     esp, 8
.text:0040109A                 cmp     esi, 5
.text:0040109D                 jl      short loc_401050 ; showStatic函数
.text:0040109F                 pop     edi
.text:004010A0                 xor     eax, eax
.text:004010A2                 pop     esi
.text:004010A3                 retn
.text:004010A3 _main           endp
```

7.3.堆变量

堆变量

- 使用malloc和new申请堆空间，返回的数据是申请的堆空间地址
- 使用free和delete释放堆空间

release

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


int main(int argc, char* argv[]) {
    char* buffer1 = (char*)malloc(10);  // 申请堆空间
    char* buffer2 = new char[10];   // 申请堆空间
    if (buffer2 != NULL) {
        delete[] buffer2;   // 释放堆空间
        buffer2 = NULL;
    }
    if (buffer1 != NULL) {
        free(buffer1);      // 释放堆空间
        buffer1 = NULL;
    }
    return 0;
}
```

ida

```
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401000
.text:00401000 argc            = dword ptr  8
.text:00401000 argv            = dword ptr  0Ch
.text:00401000
.text:00401000                 push    esi
.text:00401001                 push    10              ; Size
.text:00401003                 call    ds:__imp__malloc
.text:00401009                 push    10              ; size
.text:0040100B                 mov     esi, eax
.text:0040100D                 call    ??_U@YAPAXI@Z   ; operator new[](uint)
.text:00401012                 push    eax             ; block
.text:00401013                 call    ??_V@YAXPAX@Z   ; operator delete[](void *)
.text:00401018                 add     esp, 0Ch
.text:0040101B                 test    esi, esi
.text:0040101D                 jz      short loc_401029
.text:0040101F                 push    esi             ; Block
.text:00401020                 call    ds:__imp__free
.text:00401026                 add     esp, 4
.text:00401029
.text:00401029 loc_401029:                             ; CODE XREF: _main+1D↑j
.text:00401029                 xor     eax, eax
.text:0040102B                 pop     esi
.text:0040102C                 retn
.text:0040102C _main           endp
```

八、数组和指针的寻址

8.1.数组在函数内

在函数内定义数组

- 去其它声明，该数组即为局部变量，拥有局部变量的所有特性
- 数组名称表示该数组的首地址
- 占用的内存空间大小为：sizeof(数据类型)x数组中元素个数

- 数组的各元素应为同一数据类型，以此可以区分局部变量与数组

字符数组初始化为字符串

release

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


int main(int argc, char* argv[]) {
    char s[] = "Hello World!";
    printf("%s",s);

    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 s               = byte ptr -14h
.text:00401040 var_4           = dword ptr -4
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 sub     esp, 14h
.text:00401046                 mov     eax, ___security_cookie
.text:0040104B                 xor     eax, ebp
.text:0040104D                 mov     [ebp+var_4], eax
.text:00401050                 mov     eax, ds:dword_402108 ; 四字节:  'rld!'
.text:00401055                 movq    xmm0, qword ptr ds:aHelloWo ; "Hello Wo"
.text:0040105D                 mov     dword ptr [ebp+s+8], eax
.text:00401060                 mov     al, ds:byte_40210C ; 1字节:  0
.text:00401065                 mov     [ebp+s+0Ch], al
.text:00401068                 lea     eax, [ebp+s]
.text:0040106B                 push    eax
.text:0040106C                 push    offset _Format  ; "%s"
.text:00401071                 movq    qword ptr [ebp+s], xmm0
.text:00401076                 call    _printf
.text:0040107B                 mov     ecx, [ebp+var_4]
.text:0040107E                 add     esp, 8
.text:00401081                 xor     ecx, ebp        ; StackCookie
.text:00401083                 xor     eax, eax
.text:00401085                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:0040108A                 mov     esp, ebp
.text:0040108C                 pop     ebp
.text:0040108D                 retn
.text:0040108D _main           endp
```

8.2.数组作为参数

1.**strlen()**

release

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[]) {
    return strlen(argv[0]);
}
```

ida

```
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401000
.text:00401000 argc            = dword ptr  8
.text:00401000 argv            = dword ptr  0Ch
.text:00401000 envp            = dword ptr  10h
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 mov     eax, [ebp+argv]
.text:00401006                 mov     eax, [eax]      ; 获取参数内容，eax中被赋值字符串首地址
.text:00401008                 lea     edx, [eax+1]
.text:0040100B                 nop     dword ptr [eax+eax+00h]
.text:00401010
.text:00401010 loc_401010:                             ; CODE XREF: _main+15↓j
.text:00401010                 mov     cl, [eax]       ; 获取字符
.text:00401012                 inc     eax             ; 获取下一个字符
.text:00401013                 test    cl, cl
.text:00401015                 jnz     short loc_401010 ; 如果字符是'\0'，结束循环
.text:00401017                 sub     eax, edx        ; 字符串结束地址-字符串起止地址=字符串长度
.text:00401019                 pop     ebp
.text:0040101A                 retn
.text:0040101A _main           endp
```

2.strcpy()

在字符串初始化时，利用xmm寄存器初始化数组的值，一次可以初始化16字节，效率更高

release版

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[]) {
    char buffer[20] = { 0 }; //字符数组定义
    strcpy(buffer, argv[0]); //字符串复制
    printf(buffer);
    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 buffer          = byte ptr -18h
.text:00401040 var_4           = dword ptr -4
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 sub     esp, 18h
.text:00401046                 mov     eax, ___security_cookie
.text:0040104B                 xor     eax, ebp
.text:0040104D                 mov     [ebp+var_4], eax
.text:00401050                 mov     eax, [ebp+argv] ; eax = argv
.text:00401053                 lea     edx, [ebp+buffer] ; edx = buffer
.text:00401056                 xorps   xmm0, xmm0      ; xmm0 = 0
.text:00401059                 mov     dword ptr [ebp+buffer+10h], 0 ; buff最后4字节初始化为0
.text:00401060                 movups  xmmword ptr [ebp+buffer], xmm0 ; buff前16字节初始化为0
.text:00401064                 mov     eax, [eax]      ; eax = argv[0]
.text:00401066                 sub     edx, eax        ; edx保存两个缓冲区地址差值
.text:00401068                 nop     dword ptr [eax+eax+00000000h]
.text:00401070
.text:00401070 loc_401070:                             ; CODE XREF: _main+3B↓j
.text:00401070                 mov     cl, [eax]       ; 取出argv的字符
.text:00401072                 lea     eax, [eax+1]    ; 指向下一个字符的地址
.text:00401075                 mov     [edx+eax-1], cl ; 复制字符,通过argv[0]的地址加上差值算出buffer的地址
.text:00401079                 test    cl, cl
.text:0040107B                 jnz     short loc_401070 ; 取出argv的字符
.text:0040107D                 lea     eax, [ebp+buffer]
.text:00401080                 push    eax             ; _Format
.text:00401081                 call    _printf
.text:00401086                 mov     ecx, [ebp+var_4]
.text:00401089                 add     esp, 4
.text:0040108C                 xor     ecx, ebp        ; StackCookie
.text:0040108E                 xor     eax, eax
.text:00401090                 call    @__security_check_cookie@4 ; __security_check_cookie(x)
.text:00401095                 mov     esp, ebp
.text:00401097                 pop     ebp
.text:00401098                 retn
.text:00401098 _main           endp
```

8.3.存放指针类型数组的数组

release

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

int main(int argc, char* argv[]) {
    const char* ary[3] = { "Hello ", "World ", "!\n" };//字符串指针数组定义
        for (int i = 0; i < 3; i++) {
            printf(ary[i]); //显示输出字符串数组中的各项
        }
    return 0;
}
```

ida

```
.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401040
.text:00401040 ary             = dword ptr -0Ch
.text:00401040 argc            = dword ptr  8
.text:00401040 argv            = dword ptr  0Ch
.text:00401040 envp            = dword ptr  10h
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 sub     esp, 0Ch
.text:00401046                 push    esi
.text:00401047                 mov     [ebp+ary], offset aHello ; "Hello "
.text:0040104E                 xor     esi, esi
.text:00401050                 mov     [ebp+ary+4], offset aWorld ; "World "
.text:00401057                 mov     [ebp+ary+8], offset asc_402110 ; "!\n"
.text:0040105E                 xchg    ax, ax
.text:00401060
.text:00401060 loc_401060:                             ; CODE XREF: _main+30↓j
.text:00401060                 push    [ebp+esi*4+ary] ; _Format
.text:00401064                 call    _printf
.text:00401069                 inc     esi
.text:0040106A                 add     esp, 4
.text:0040106D                 cmp     esi, 3
.text:00401070                 jl      short loc_401060
.text:00401072                 xor     eax, eax
.text:00401074                 pop     esi
.text:00401075                 mov     esp, ebp
.text:00401077                 pop     ebp
.text:00401078                 retn
.text:00401078 _main           endp
```

8.4.函数指针

release

```c
#include <stdio.h>

int _stdcall show(int n) { //函数定义
    printf("show : %d\n", n);
    return n;
}
int main(int argc, char* argv[]) {
    int(_stdcall * pfn)(int) = show; //函数指针定义并初始化
    int ret = pfn(5); //使用函数指针调用函数并获取返回值
    printf("ret = %d\n", ret);
    return 0;
}
```

ida

```
.text:00401060 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401060 _main           proc near               ; CODE XREF: __scrt_common_main_seh+F5↓p
.text:00401060
.text:00401060 argc            = dword ptr  8
.text:00401060 argv            = dword ptr  0Ch
.text:00401060
.text:00401060                 push    5               ; n
.text:00401062                 call    ?show@@YGHH@Z   ; show(int)
.text:00401067                 push    eax
.text:00401068                 push    offset aRetD    ; "ret = %d\n"
.text:0040106D                 call    _printf
.text:00401072                 add     esp, 8
.text:00401075                 xor     eax, eax
.text:00401077                 retn
.text:00401077 _main           endp
```

九、结构体和类

9.1.对象的内存布局

1.空类

- 空类的长度位1字节

2.内存对齐

- 结构体中的数据成员类型最大值为M，指定对齐值为N，则实际对齐值为q=min(M,N)

3.静态数据成员

- 类中的数据成员被修饰为静态时，它与局部静态变量类似，存放的位置和全局变量一致

9.2.this指针

对象调用成员的方法以及取出数据成员的过程

- 利用寄存器ecx保存对象的首地址
- 以寄存器传参的方式将其传递到成员函数中

debug版

```
#include <stdio.h>

class Person {
public:
    void setAge(int age) { //公有成员函数
        this->age = age;
    }
public:
    int age; //公有数据成员
};
int main(int argc, char* argv[]) {
    Person person;
    person.setAge(5); //调用成员函数
    printf("Person : %d\n", person.age); //获取数据成员
    return 0;
}
```

ida

```
.text:0045485B __$EncStackInitEnd_2:
.text:0045485B                 mov     eax, ___security_cookie
.text:00454860                 xor     eax, ebp
.text:00454862                 mov     [ebp+var_4], eax
.text:00454865                 mov     ecx, offset _8E097BDB_main@cpp ; JMC_flag
.text:0045486A                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045486F                 push    5               ; age
.text:00454871                 lea     ecx, [ebp+person] ; this
.text:00454874                 call    j_?setAge@Person@@QAEXH@Z ; Person::setAge(int)
.text:00454879                 mov     eax, [ebp+person.age]
.text:0045487C                 push    eax
.text:0045487D                 push    offset _Format  ; "Person : %d\n"
.text:00454882                 call    j__printf
.text:00454887                 add     esp, 8
.text:0045488A                 xor     eax, eax
```

```
.text:0045472C __$EncStackInitEnd:
.text:0045472C                 pop     ecx
.text:0045472D                 mov     [ebp+this], ecx ; ecx保存person对象的首地址
.text:00454730                 mov     ecx, offset _8E097BDB_main@cpp ; JMC_flag
.text:00454735                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045473A                 mov     eax, [ebp+this] ; eax = this
.text:0045473D                 mov     ecx, [ebp+age]  ; ecx = age
.text:00454740                 mov     [eax], ecx      ; this->age = age
.text:00454742                 pop     edi
.text:00454743                 pop     esi
.text:00454744                 pop     ebx
.text:00454745                 add     esp, 0CCh
.text:0045474B                 cmp     ebp, esp
.text:0045474D                 call    j___RTC_CheckEsp
.text:00454752                 mov     esp, ebp
.text:00454754                 pop     ebp
.text:00454755                 retn    4
.text:00454755 ?setAge@Person@@QAEXH@Z endp
```

9.3.对象作为函数参数

debug

```
#include <stdio.h>

class Person {
public:
    int age;
    int height;
};

void show(Person person) { //参数为类Person的对象
    printf("age = %d , height = %d\n", person.age,person.height);
}

int main(int argc, char* argv[]) {
    Person person;
    person.age = 1;
    person.height = 2;
    show(person);
    return 0;
}
```

ida

```
.text:00454865                 mov     [ebp+var_C], 1
.text:0045486C                 mov     [ebp+var_8], 2
.text:00454873                 mov     eax, [ebp+var_8]
.text:00454876                 push    eax
.text:00454877                 mov     ecx, [ebp+var_C]
.text:0045487A                 push    ecx
.text:0045487B                 call    sub_44E6C2      ; 调用show函数
```

```
.text:00454731                 mov     eax, [ebp+arg_4]
.text:00454734                 push    eax
.text:00454735                 mov     ecx, [ebp+arg_0]
.text:00454738                 push    ecx
.text:00454739                 push    offset aAgeDHeightD ; "age = %d , height = %d\n"
.text:0045473E                 call    sub_44EA50
.text:00454743                 add     esp, 0Ch
.text:00454746                 pop     edi
.text:00454747                 pop     esi
```

含有数组数据成员的对象传参

debug

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

class Person {
public:
    int age;
    int height;
    char name[32]; //定义数组类型的数据成员
};
void show(Person person) {
    printf("age = %d , height = %d name:%s\n", person.age,
        person.height, person.name);
}
int main(int argc, char* argv[]) {
    Person person;
    person.age = 1;
    person.height = 2;
    strcpy(person.name, "tom"); //赋值数据成员数组
    show(person);
    return 0;
}
```

ida

```
.text:00454860 ; int __cdecl main_0(int argc, const char **argv, const char **envp)
.text:00454860 _main_0         proc near               ; CODE XREF: _main↑j
.text:00454860
.text:00454860 var_34          = byte ptr -34h
.text:00454860 var_30          = dword ptr -30h
.text:00454860 var_2C          = dword ptr -2Ch
.text:00454860 Destination     = byte ptr -28h
.text:00454860 var_4           = dword ptr -4
.text:00454860 argc            = dword ptr  8
.text:00454860 argv            = dword ptr  0Ch
.text:00454860 envp            = dword ptr  10h
.text:00454860
.text:00454860                 push    ebp
.text:00454861                 mov     ebp, esp
.text:00454863                 sub     esp, 0F4h
.text:00454869                 push    ebx
.text:0045486A                 push    esi
.text:0045486B                 push    edi
.text:0045486C                 lea     edi, [ebp+var_34]
.text:0045486F                 mov     ecx, 0Dh
.text:00454874                 mov     eax, 0CCCCCCCCh
.text:00454879                 rep stosd
.text:0045487B                 mov     eax, ___security_cookie
.text:00454880                 xor     eax, ebp
.text:00454882                 mov     [ebp+var_4], eax
.text:00454885                 mov     ecx, offset unk_51C007
.text:0045488A                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045488F                 mov     [ebp+var_30], 1 ; person.age = 1，对象的首地址
.text:00454896                 mov     [ebp+var_2C], 2 ; person.height = 2
.text:0045489D                 push    offset Source   ; "tom"
.text:004548A2                 lea     eax, [ebp+Destination]
.text:004548A5                 push    eax             ; Destination
.text:004548A6                 call    j__strcpy
.text:004548AB                 add     esp, 8
.text:004548AE                 sub     esp, 28h
.text:004548B1                 mov     ecx, 0Ah        ; 设置循环次数为10
.text:004548B6                 lea     esi, [ebp+var_30] ; esi保存对象的首地址
.text:004548B9                 mov     edi, esp        ; edi为当前栈顶
.text:004548BB                 rep movsd               ; 执行10次4字节内存复制，将esi所指向的数据复制到edi中
.text:004548BD                 call    sub_44E6C2      ; 调用show函数
.text:004548C2                 add     esp, 28h
.text:004548C5                 xor     eax, eax
```

9.4.对象作为返回值

debug

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

class Person {
public:
    int count;
    int buffer[10]; //定义两个数据成员，该类的大小为44字节
};

Person getPerson() {
    Person person;
    person.count = 10;
    for (int i = 0; i < 10; i++) {
        person.buffer[i] = i + 1;
    }
    return person;
}

int main(int argc, char* argv[]) {
    Person person;
    person = getPerson();
    printf("%d %d %d", person.count, person.buffer[0],person.buffer[9]);
    return 0;
}
```

ida

```
.text:004548B0                push    ebp
.text:004548B1                mov     ebp, esp
.text:004548B3                sub     esp, 15Ch
.text:004548B9                push    ebx
.text:004548BA                push    esi
.text:004548BB                push    edi
.text:004548BC                lea     edi, [ebp+var_9C]
.text:004548C2                mov     ecx, 27h
.text:004548C7                mov     eax, 0CCCCCCCCh
.text:004548CC                rep stosd
.text:004548CE                mov     ecx, offset unk_51C007
.text:004548D3                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:004548D8                lea     eax, [ebp+var_158] ; 获取返回对象的栈空间首地址
.text:004548DE                push    eax             ; 将返回对象的首地址压入栈中，用于保存返回对象的数据
.text:004548DF                call    sub_44FF2C      ; 调用getPerson函数
.text:004548E4                add     esp, 4
.text:004548E7                mov     ecx, 0Bh        ; 设置循环次数11
.text:004548EC                mov     esi, eax        ; 将返回对象的首地址存入esi中
.text:004548EE                lea     edi, [ebp+var_124] ; 获取临时对象的首地址
.text:004548F4                rep movsd               ; 每次从返回对象中复制4字节数据到临时对象的地址中，共11次
.text:004548F6                mov     ecx, 0Bh        ; 重新设置复制次数为11
.text:004548FB                lea     esi, [ebp+var_124] ; 获取临时对象的首地址
.text:00454901                lea     edi, [ebp+var_30] ; 获取对象person的首地址
.text:00454904                rep movsd               ; 将数据复制到对象person中
.text:00454906                mov     eax, 4
.text:0045490B                imul    ecx, eax, 9
.text:0045490E                mov     edx, [ebp+ecx+var_2C] ; person.buffer[9]
.text:00454912                push    edx
.text:00454913                mov     eax, 4
.text:00454918                imul    ecx, eax, 0     ; person.buffer[0]
.text:0045491B                mov     edx, [ebp+ecx+var_2C]
.text:0045491F                push    edx
.text:00454920                mov     eax, [ebp+var_30] ; person.count
.text:00454923                push    eax
.text:00454924                push    offset aDDD     ; "%d %d %d"
.text:00454929                call    sub_44EA4B
.text:0045492E                add     esp, 10h
.text:00454931                xor     eax, eax
```

getperson函数

```
.text:00454710                push    ebp
.text:00454711                mov     ebp, esp
.text:00454713                sub     esp, 100h
.text:00454719                push    ebx
.text:0045471A                push    esi
.text:0045471B                push    edi
.text:0045471C                lea     edi, [ebp+var_40]
.text:0045471F                mov     ecx, 10h
.text:00454724                mov     eax, 0CCCCCCCCh
.text:00454729                rep stosd
.text:0045472B                mov     ecx, offset unk_51C007
.text:00454730                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:00454735                mov     [ebp+var_30], 0Ah ; person.count = 10
.text:0045473C                mov     [ebp+var_3C], 0 ; i = 0
.text:00454743                jmp     short loc_45474E ; for循环
.text:00454745 ; ---------------------------------------------------------------------------
.text:00454745
.text:00454745 loc_454745:                            ; CODE XREF: sub_454710+51↓j
.text:00454745                mov     eax, [ebp+var_3C]
.text:00454748                add     eax, 1
.text:0045474B                mov     [ebp+var_3C], eax
.text:0045474E
.text:0045474E loc_45474E:                            ; CODE XREF: sub_454710+33↑j
.text:0045474E                cmp     [ebp+var_3C], 0Ah
.text:00454752                jge     short loc_454763 ; i>10，循环结束
.text:00454754                mov     eax, [ebp+var_3C]
.text:00454757                add     eax, 1
.text:0045475A                mov     ecx, [ebp+var_3C]
.text:0045475D                mov     [ebp+ecx*4+var_2C], eax ; person.buffer[i] = i+1
.text:00454761                jmp     short loc_454745
.text:00454763 ; ---------------------------------------------------------------------------
.text:00454763
.text:00454763 loc_454763:                            ; CODE XREF: sub_454710+42↑j
.text:00454763                mov     ecx, 0Bh        ; 设置循环次数11次
.text:00454768                lea     esi, [ebp+var_30] ; 获取局部对象的首地址，&person
.text:0045476B                mov     edi, [ebp+arg_0] ; 获取返回对象的首地址
.text:0045476E                rep movsd               ; 将局部对象person中的数据复制到返回对象中
.text:00454770                mov     eax, [ebp+arg_0] ; 获取返回对象的首地址并保存到eax中，作为返回值
.text:00454773                push    edx
.text:00454774                mov     ecx, ebp        ; Esp
.text:00454776                push    eax
.text:00454777                lea     edx, Fd         ; Fd
.text:0045477D                call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
```

十、构造函数和析构函数

根据生命周期将对象进行分类，分析各类对象构造函数和析构函数的调用时机

- 局部对象
- 堆对象
- 参数对象
- 返回对象
- 全局对象
- 静态对象

10.1.构造函数的出现时机

10.1.1.局部对象

debug

```
#include <stdio.h>

class Person {
public:
    Person() { //无参构造函数
        age = 20;
    }
    int age;
};
```

```
int main(int argc, char* argv[]) {
    Person person; //类对象定义
    return 0;

}
```

ida

```
.text:00453770                push    ebp
.text:00453771                mov     ebp, esp
.text:00453773                sub     esp, 0D0h
.text:00453779                push    ebx
.text:0045377A                push    esi
.text:0045377B                push    edi
.text:0045377C                lea     edi, [ebp+var_10]
.text:0045377F                mov     ecx, 4
.text:00453784                mov     eax, 0CCCCCCCCh
.text:00453789                rep stosd
.text:0045378B                mov     eax, ___security_cookie
.text:00453790                xor     eax, ebp
.text:00453792                mov     [ebp+var_4], eax
.text:00453795                mov     ecx, offset unk_51B003
.text:0045379A                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045379F                lea     ecx, [ebp+var_C] ; ecx = 对象首地址，&person
.text:004537A2                call    sub_450016      ; 调用构造函数
.text:004537A7                xor     eax, eax
.text:004537A9                push    edx
.text:004537AA                mov     ecx, ebp        ; Esp
.text:004537AC                push    eax
.text:004537AD                lea     edx, Fd         ; Fd
.text:004537B3                call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
```

构造函数

```
.text:00453700 var_C           = byte ptr -0Ch
.text:00453700 var_8           = dword ptr -8
.text:00453700
.text:00453700                 push    ebp
.text:00453701                 mov     ebp, esp
.text:00453703                 sub     esp, 0CCh
.text:00453709                 push    ebx
.text:0045370A                 push    esi
.text:0045370B                 push    edi
.text:0045370C                 push    ecx
.text:0045370D                 lea     edi, [ebp+var_C]
.text:00453710                 mov     ecx, 3
.text:00453715                 mov     eax, 0CCCCCCCCh
.text:0045371A                 rep stosd
.text:0045371C                 pop     ecx
.text:0045371D                 mov     [ebp+var_8], ecx ; this指针
.text:00453720                 mov     ecx, offset unk_51B003
.text:00453725                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045372A                 mov     eax, [ebp+var_8] ; eax保存对象的首地址
.text:0045372D                 mov     dword ptr [eax], 14h ; 将数据成员age设置为20
.text:00453733                 mov     eax, [ebp+var_8] ; 将this指针存入eax，作为返回值
.text:00453736                 pop     edi
.text:00453737                 pop     esi
.text:00453738                 pop     ebx
.text:00453739                 add     esp, 0CCh
.text:0045373F                 cmp     ebp, esp
.text:00453741                 call    j___RTC_CheckEsp
.text:00453746                 mov     esp, ebp
.text:00453748                 pop     ebp
.text:00453749                 retn
.text:00453749 sub_453700      endp
```

总结：局部对象构造函数的必要条件

- 该成员函数是这个对象在作用域内调用的第一个成员函数，根据this指针可以区分每个对象
- 这个成员函数通过thiscall方式调用
- 这个函数返回this指针

10.1.2堆对象

debug

```
#include <stdio.h>

class Person {
public:
    Person() {
        age = 20;
    }
    int age;
};

int main(int argc, char* argv[]) {
    Person* p = new Person;
    //为了突出本节讨论的问题，这里没有检查new运算的返回值
    p->age = 21;
    printf("%d\n", p->age);

    return 0;

}
```

ida

```
.text:004548E8                 push    esi
.text:004548E9                 push    edi
.text:004548EA                 lea     edi, [ebp+var_34]
.text:004548ED                 mov     ecx, 0Ah
.text:004548F2                 mov     eax, 0CCCCCCCCh
.text:004548F7                 rep stosd
.text:004548F9                 mov     eax, ___security_cookie
.text:004548FE                 xor     eax, ebp
.text:00454900                 push    eax
.text:00454901                 lea     eax, [ebp+var_C]
.text:00454904                 mov     large fs:0, eax
.text:0045490A                 mov     ecx, offset unk_51D003
.text:0045490F                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:00454914                 push    4               ; 压入类的大小，用于堆内存申请
.text:00454916                 call    j_??2@YAPAXI@Z  ; 调用new函数
.text:0045491B                 add     esp, 4
.text:0045491E                 mov     [ebp+var_EC], eax ; 使用临时变量保存new返回值
.text:00454924                 mov     [ebp+var_4], 0
.text:0045492B                 cmp     [ebp+var_EC], 0 ; 检测堆内存是否申请成功
.text:00454932                 jz      short loc_454947 ; 申请失败则跳过构造函数调用
.text:00454934                 mov     ecx, [ebp+var_EC] ; 如果申请成功，将对象首地址传给ecx
.text:0045493A                 call    sub_45105C      ; 调用构造函数
.text:0045493F                 mov     [ebp+var_F4], eax ; 构造函数返回this指针，保存到临时变量
.text:00454945                 jmp     short loc_454951
.text:00454947 ; ---------------------------------------------------------------------------
.text:00454947
.text:00454947 loc_454947:                             ; CODE XREF: _main_0+62↑j
.text:00454947                 mov     [ebp+var_F4], 0 ; 申请堆空间失败，设置指针值为NULL
.text:00454951
.text:00454951 loc_454951:                             ; CODE XREF: _main_0+75↑j
.text:00454951                 mov     eax, [ebp+var_F4]
.text:00454957                 mov     [ebp+var_E0], eax
.text:0045495D                 mov     [ebp+var_4], 0FFFFFFFFh
.text:00454964                 mov     ecx, [ebp+var_E0] ; 指针变量P
.text:0045496A                 mov     [ebp+var_14], ecx
.text:0045496D                 mov     eax, [ebp+var_14]
.text:00454970                 mov     dword ptr [eax], 21 ; 赋值 age = 21
.text:00454976                 mov     eax, [ebp+var_14]
.text:00454979                 mov     ecx, [eax]
.text:0045497B                 push    ecx
.text:0045497C                 push    offset unk_4F3E50
.text:00454981                 call    sub_44EA5F
```

总结：

- 使用new申请堆空间之后，需要调用构造函数来完成对象数据成员的初始化
- 如果堆空间申请失败，则不调用构造函数

- 如果new执行成功，返回值是对象的首地址
- 识别堆对象的构造函数：重点分析new的双分支结构，在判定new成功的分支迅速定位并得到构造函数

10.1.3.参数对象

当对象作为函数参数时，会调用赋值构造函数

debug

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

class Person {
public:
    Person() {
        name = NULL;//无参构造函数，初始化指针
    }

    Person(const Person& obj) {
        // 注：如果在复制构造函数中直接复制指针值，那么对象内的两个成员指针会指向同一个资源，这属于浅拷贝
            // this->name = obj.name;
            // 为实参对象中的指针所指向的堆空间制作一份副本，这就是深拷贝了
        int len = strlen(obj.name);
        this->name = new char[len + sizeof(char)]; // 为便于讲解，这里没有检查指针
        strcpy(this->name, obj.name);
    }

    void setName(const char* name) {
        int len = strlen(name);
        if (this->name != NULL) {
            delete[] this->name;
        }
        this->name = new char[len + sizeof(char)]; // 为便于讲解，这里没有检查指针
        strcpy(this->name, name);
    }

public:
    char* name;
};

void show(Person person) { // 参数是对象类型，会触发复制构造函数
    printf("name:%s\n", person.name);
}

int main(int argc, char* argv[]) {
    Person person;
    person.setName("Hello");
    show(person);
    return 0;
}
```

ida

```
.text:00454B60                push    ebp
.text:00454B61                mov     ebp, esp
.text:00454B63                sub     esp, 0DCh
.text:00454B69                push    ebx
.text:00454B6A                push    esi
.text:00454B6B                push    edi
.text:00454B6C                lea     edi, [ebp+var_1C]
.text:00454B6F                mov     ecx, 7
.text:00454B74                mov     eax, 0CCCCCCCCh
.text:00454B79                rep stosd
.text:00454B7B                mov     eax, ___security_cookie
.text:00454B80                xor     eax, ebp
.text:00454B82                mov     [ebp+var_4], eax
.text:00454B85                mov     ecx, offset unk_51D007
.text:00454B8A                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:00454B8F                push    4               ; Size
.text:00454B91                lea     ecx, [ebp+var_C]
.text:00454B94                call    sub_4503B9
.text:00454B99                lea     ecx, [ebp+var_C] ; ecx = &person
.text:00454B9C                call    sub_45107F      ; 调用构造函数
.text:00454BA1                push    offset aHello   ; "Hello"
.text:00454BA6                lea     ecx, [ebp+var_C] ; ecx = &person
.text:00454BA9                call    sub_450B4D      ; 调用成员函数setName
.text:00454BAE                push    ecx             ; 这里的push ecx等价于suu esp 4
.text:00454BAF                mov     ecx, esp        ; 获取参数对象的地址，保存到ecx中
.text:00454BB1                lea     eax, [ebp+var_C] ; 获取对象person的地址
.text:00454BB4                push    eax             ; 将person地址作为参数
.text:00454BB5                call    sub_45076A      ; 调用复制构造函数
.text:00454BBA                call    sub_44E6CC      ; 调用show函数
.text:00454BBF                add     esp, 4
.text:00454BC2                xor     eax, eax
.text:00454BC4                push    edx
.text:00454BC5                mov     ecx, ebp        ; Esp
.text:00454BC7                push    eax
.text:00454BC8                lea     edx, Fd         ; Fd
.text:00454BCE                call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
.text:00454BD3                pop     eax
.text:00454BD4                pop     edx
.text:00454BD5                pop     edi
.text:00454BD6                pop     esi
.text:00454BD7                pop     ebx
.text:00454BD8                mov     ecx, [ebp+var_4]
```

调用赋值构造函数

```
.text:004547F0                push    ebp
.text:004547F1                mov     ebp, esp
.text:004547F3                sub     esp, 0E4h
.text:004547F9                push    ebx
.text:004547FA                push    esi
.text:004547FB                push    edi
.text:004547FC                push    ecx
.text:004547FD                lea     edi, [ebp+var_24]
.text:00454800                mov     ecx, 9
.text:00454805                mov     eax, 0CCCCCCCCh
.text:0045480A                rep stosd
.text:0045480C                pop     ecx
.text:0045480D                mov     [ebp+var_8], ecx
.text:00454810                mov     ecx, offset unk_51D007
.text:00454815                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045481A                mov     eax, [ebp+arg_0] ; eax = &obj
.text:0045481D                mov     ecx, [eax]       ; ecx = obj.name
.text:0045481F                push    ecx              ; 参数1
.text:00454820                call    j__strlen        ; 调用strlen函数
.text:00454825                add     esp, 4
.text:00454828                mov     [ebp+var_14], eax ; len = strlen(obj.name)
.text:0045482B                mov     eax, [ebp+var_14]
.text:0045482E                add     eax, 1
.text:00454831                push    eax              ; Size
.text:00454832                call    j_unknown_libname_42 ; 调用new函数
.text:00454837                add     esp, 4
.text:0045483A                mov     [ebp+var_E0], eax
.text:00454840                mov     ecx, [ebp+var_8]
.text:00454843                mov     edx, [ebp+var_E0]
.text:00454849                mov     [ecx], edx       ; this->name = new char[len + sizeof(char)]
.text:0045484B                mov     eax, [ebp+arg_0]
.text:0045484E                mov     ecx, [eax]
.text:00454850                push    ecx              ; Source    obj.name
.text:00454851                mov     edx, [ebp+var_8]
.text:00454854                mov     eax, [edx]
.text:00454856                push    eax              ; Destination    this->name
.text:00454857                call    j__strcpy
.text:0045485C                add     esp, 8
.text:0045485F                mov     eax, [ebp+var_8] ; 返回this指针
.text:00454862                pop     edi
.text:00454863                pop     esi
.text:00454864                pop     ebx
.text:00454865                add     esp, 0E4h
```

10.4.返回对象

返回对象与参数对象类似，都会使用赋值构造函数。但是，两者使用时机不同

- 当对象为参数时，在进入函数前使用赋值构造函数
- 返回对象则在函数返回时使用赋值构造函数

debug

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

class Person {
public:
    Person() {
        name = NULL;//无参构造函数，初始化指针
    }
    Person(const Person& obj) {
        // 注：如果在复制构造函数中直接复制指针值，那么对象内的两个成员指针会指向同一个资源，这属于浅拷贝
            // this->name = obj.name;
            // 为实参对象中的指针所指向的堆空间制作一份副本，这就是深拷贝了
        int len = strlen(obj.name);
        this->name = new char[len + sizeof(char)]; // 为便于讲解，这里没有检查指针
        strcpy(this->name, obj.name);
    }
    void setName(const char* name) {
        int len = strlen(name);
```

```
        if (this->name != NULL) {
            delete[] this->name;
        }
        this->name = new char[len + sizeof(char)]; // 为便于讲解，这里没有检查指针
        strcpy(this->name, name);
    }
public:
    char* name;
};

Person getObject() {
    Person person;
    person.setName("Hello");
    return person; //返回类型为对象
}
int main(int argc, char* argv[]) {
    Person person = getObject();
    return 0;
}
```

ida

```
.text:00454AF0
.text:00454AF0                push    ebp
.text:00454AF1                mov     ebp, esp
.text:00454AF3                sub     esp, 0D0h
.text:00454AF9                push    ebx
.text:00454AFA                push    esi
.text:00454AFB                push    edi
.text:00454AFC                lea     edi, [ebp+var_10]
.text:00454AFF                mov     ecx, 4
.text:00454B04                mov     eax, 0CCCCCCCCh
.text:00454B09                rep stosd
.text:00454B0B                mov     eax, ___security_cookie
.text:00454B10                xor     eax, ebp
.text:00454B12                mov     [ebp+var_4], eax
.text:00454B15                mov     ecx, offset unk_51D007
.text:00454B1A                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:00454B1F                push    4               ; Size
.text:00454B21                lea     ecx, [ebp+var_C]
.text:00454B24                call    sub_4503AF
.text:00454B29                lea     eax, [ebp+var_C] ; 去对象person的首地址
.text:00454B2C                push    eax             ; 将对象的首地址作为参数传递
.text:00454B2D                call    sub_450B93      ; 调用getObject函数
.text:00454B32                add     esp, 4
.text:00454B35                xor     eax, eax
.text:00454B37                push    edx
.text:00454B38                mov     ecx, ebp        ; Esp
.text:00454B3A                push    eax
.text:00454B3B                lea     edx, stru_454B68 ; Fd
.text:00454B41                call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
.text:00454B46                pop     eax
```

调用getObject函数

```
.text:00454940                push    ebp
.text:00454941                mov     ebp, esp
.text:00454943                sub     esp, 0D0h
.text:00454949                push    ebx
.text:0045494A                push    esi
.text:0045494B                push    edi
.text:0045494C                lea     edi, [ebp+var_10]
.text:0045494F                mov     ecx, 4
.text:00454954                mov     eax, 0CCCCCCCCh
.text:00454959                rep stosd
.text:0045495B                mov     eax, ___security_cookie
.text:00454960                xor     eax, ebp
.text:00454962                mov     [ebp+var_4], eax
.text:00454965                mov     ecx, offset unk_51D007
.text:0045496A                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045496F                push    4               ; Size
.text:00454971                lea     ecx, [ebp+var_C]
.text:00454974                call    sub_4503AF
.text:00454979                lea     ecx, [ebp+var_C] ; 将局部对象的首地址作为参数传递
.text:0045497C                call    sub_45107A      ; 调用构造函数
.text:00454981                push    offset aHello   ; "Hello"
.text:00454986                lea     ecx, [ebp+var_C]
.text:00454989                call    sub_450B43      ; 调用成员函数setName
.text:0045498E                lea     eax, [ebp+var_C] ; 获取局部对象的首地址
.text:00454991                push    eax
.text:00454992                mov     ecx, [ebp+arg_0] ; 获取参数中保存的this指针
.text:00454995                call    sub_450760      ; 调用赋值构造函数
.text:0045499A                mov     eax, [ebp+arg_0] ; 将局部对象的首地址作为参数传递
.text:0045499D                push    edx
.text:0045499E                mov     ecx, ebp        ; Esp
.text:004549A0                push    eax
.text:004549A1                lea     edx, Fd         ; Fd
.text:004549A7                call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
.text:004549AC                pop     eax
.text:004549AD                pop     edx
.text:004549AE                pop     edi
.text:004549AF                pop     esi
.text:004549B0                pop     ebx
```

10.2.析构对象的出现时机

10.2.1.局部对象

重点考察作用域的结束处，当对象所在作用域结束后，将销毁作用域所有变量的栈空间，此时便是析构函数出现的时机。析构函数同样属于成员函数，因此在调用的过程中也需要传递this指针。

debug

```
#include <stdio.h>

class Person {
public:
    Person() {
        age = 1;
    }
    ~Person() {
        printf("~Person()\n");
    }
private:
    int age;
};
int main(int argc, char* argv[]) {
    Person person;
    return 0; //退出函数后调用析构函数
}
```

ida

```
.text:004548B0                push    ebp
.text:004548B1                mov     ebp, esp
.text:004548B3                sub     esp, 0DCh
.text:004548B9                push    ebx
.text:004548BA                push    esi
.text:004548BB                push    edi
.text:004548BC                lea     edi, [ebp+var_1C]
.text:004548BF                mov     ecx, 7
.text:004548C4                mov     eax, 0CCCCCCCCh
.text:004548C9                rep stosd
.text:004548CB                mov     eax, ___security_cookie
.text:004548D0                xor     eax, ebp
.text:004548D2                mov     [ebp+var_4], eax
.text:004548D5                mov     ecx, offset unk_51C003
.text:004548DA                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:004548DF                lea     ecx, [ebp+var_C] ; 获取对象的首地址，作为this指针
.text:004548E2                call    sub_451020       ; 调用构造函数
.text:004548E7                mov     [ebp+var_D8], 0
.text:004548F1                lea     ecx, [ebp+var_C]
.text:004548F4                call    sub_44EA6E       ; 调用析构函数
.text:004548F9                mov     eax, [ebp+var_D8]
.text:004548FF                push    edx
.text:00454900                mov     ecx, ebp         ; Esp
.text:00454902                push    eax
.text:00454903                lea     edx, Fd          ; Fd
.text:00454909                call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
.text:0045490E                pop     eax
.text:0045490F                pop     edx
```

10.2.2.堆对象

用detele释放对象所在的空间，delete的使用便是找到堆对象调用析构的关键点

debug

```
#include <stdio.h>

class Person {
public:
    Person() {
        age = 20;
    }
    ~Person() {
        printf("~Person()\n");
    }
    int age;
};
int main(int argc, char* argv[]) {
    Person* person = new Person();
    person->age = 21; //为了便于讲解，这里没检查指针
    printf("%d\n", person->age);
    delete person;
    return 0;
}
```

ida

```
.text:004549EF                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:004549F4                push    4               ; Size
.text:004549F6                call    j_??2@YAPAXI@Z  ; 调用new函数
.text:004549FB                add     esp, 4
.text:004549FE                mov     [ebp+var_EC], eax ; 保存申请的内存地址
.text:00454A04                mov     [ebp+var_4], 0
.text:00454A0B                cmp     [ebp+var_EC], 0
.text:00454A12                jz      short loc_454A27 ; 检查内存空间是否申请成功
.text:00454A14                mov     ecx, [ebp+var_EC] ; 传递this指针
.text:00454A1A                call    sub_451066      ; 调用构造函数
.text:00454A1F                mov     [ebp+var_100], eax ; 保存构造函数返回值
.text:00454A25                jmp     short loc_454A31
.text:00454A27 ; ---------------------------------------------------------------------------
.text:00454A27
.text:00454A27 loc_454A27:                            ; CODE XREF: _main_0+62↑j
.text:00454A27                mov     [ebp+var_100], 0
.text:00454A31
.text:00454A31 loc_454A31:                            ; CODE XREF: _main_0+75↑j
.text:00454A31                mov     eax, [ebp+var_100]
.text:00454A37                mov     [ebp+var_E0], eax ; 保存申请的地址到指针变量person
.text:00454A3D                mov     [ebp+var_4], 0FFFFFFFFh
.text:00454A44                mov     ecx, [ebp+var_E0]
.text:00454A4A                mov     [ebp+var_14], ecx
.text:00454A4D                mov     eax, [ebp+var_14]
.text:00454A50                mov     dword ptr [eax], 15h ; person->age=21
.text:00454A56                mov     eax, [ebp+var_14]
.text:00454A59                mov     ecx, [eax]
.text:00454A5B                push    ecx
.text:00454A5C                push    offset aD       ; "%d\n"
.text:00454A61                call    sub_44EA5F
.text:00454A66                add     esp, 8
.text:00454A69                mov     eax, [ebp+var_14]
.text:00454A6C                mov     [ebp+var_F8], eax
.text:00454A72                cmp     [ebp+var_F8], 0
.text:00454A79                jz      short loc_454A90
.text:00454A7B                push    1
.text:00454A7D                mov     ecx, [ebp+var_F8] ; 传递this指针
.text:00454A83                call    sub_44F20C      ; 调用析构代理函数
.text:00454A88                mov     [ebp+var_100], eax
.text:00454A8E                jmp     short loc_454A9A
.text:00454A90 ; ---------------------------------------------------------------------------
```

析构代理函数

```
.text:00454870                push    ebp
.text:00454871                mov     ebp, esp
.text:00454873                sub     esp, 0CCh
.text:00454879                push    ebx
.text:0045487A                push    esi
.text:0045487B                push    edi
.text:0045487C                push    ecx
.text:0045487D                lea     edi, [ebp+var_C]
.text:00454880                mov     ecx, 3
.text:00454885                mov     eax, 0CCCCCCCCh
.text:0045488A                rep stosd
.text:0045488C                pop     ecx
.text:0045488D                mov     [ebp+var_8], ecx
.text:00454890                mov     ecx, [ebp+var_8] ; 传递this指针
.text:00454893                call    sub_44EA82      ; 调用析构函数
.text:00454898                mov     eax, [ebp+arg_0]
.text:0045489B                and     eax, 1
.text:0045489E                jz      short loc_4548AE ; 检查析构函数标记
.text:004548A0                push    4
.text:004548A2                mov     eax, [ebp+var_8]
.text:004548A5                push    eax
.text:004548A6                call    sub_44E5AF      ; 调用delete函数，释放堆空间
.text:004548AB                add     esp, 8
.text:004548AE
```

十一、虚函数

对于具有虚函数的类而言，构造函数和析构函数的识别过程更加简单。而且，在类中定义虚函数后，如果没有提供

构造函数，编译器会生成默认的构造函数。

对象的多态需要通过虚表和虚指针完成，虚表指针被定义在对象首地址处，因此虚函数必须作为成员函数使用。

11.1.虚函数的机制

当类中定义有虚函数，编译器会将给类中所有虚函数的首地址保存在一张地址表，这张表被称为虚函数地址表，简称虚表。同时还会在类中添加一个隐藏数据成员，称为虚表指针，该指针保存虚表的首地址，用于记录和查找虚函数。

11.1.1.默认构造函数初始化虚表指针的过程

- 没有编写构造函数时，编译器默认提供构造函数，以完成虚表指针的初始化
- 虚表中虚函数的地址排列顺序：先声明的虚函数的地址会被排列在虚表靠前的位置
- 第一个被声明的虚函数的地址在虚表的首地址处

debug

```
#include <stdio.h>

class Person {
public:
    virtual int getAge() { //虚函数定义
        return age;
    }
    virtual void setAge(int age) { //虚函数定义
        this->age = age;
    }
private:
    int age;
};

int main(int argc, char* argv[]) {
    Person person;
    //int size = sizeof(Person);
    //定义了虚函数后，因为还含有隐藏数据成员虚表指针，所以Person大小为8
    //printf("%d",size);   8
    return 0;
}
```

ida

```
.text:00454810                push    ebp
.text:00454811                mov     ebp, esp
.text:00454813                sub     esp, 0D4h
.text:00454819                push    ebx
.text:0045481A                push    esi
.text:0045481B                push    edi
.text:0045481C                lea     edi, [ebp+var_14]
.text:0045481F                mov     ecx, 5
.text:00454824                mov     eax, 0CCCCCCCCh
.text:00454829                rep stosd
.text:0045482B                mov     eax, ___security_cookie
.text:00454830                xor     eax, ebp
.text:00454832                mov     [ebp+var_4], eax
.text:00454835                mov     ecx, offset unk_51C003
.text:0045483A                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045483F                lea     ecx, [ebp+var_10] ; 获取对象首地址
.text:00454842                call    sub_451020        ; 调用构造函数，此为默认构造函数
.text:00454847                xor     eax, eax
.text:00454849                push    edx
.text:0045484A                mov     ecx, ebp          ; Esp
.text:0045484C                push    eax
.text:0045484D                lea     edx, Fd           ; Fd
.text:00454853                call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
.text:00454858                pop     eax
.text:00454859                pop     edx
.text:0045485A                pop     edi
.text:0045485B                pop     esi
.text:0045485C                pop     ebx
.text:0045485D                mov     ecx, [ebp+var_4]
.text:00454860                xor     ecx, ebp
.text:00454862                call    sub_44F27A
.text:00454867                add     esp, 0D4h
.text:0045486D                cmp     ebp, esp
.text:0045486F                call    j___RTC_CheckEsp
.text:00454874                mov     esp, ebp
.text:00454876                pop     ebp
.text:00454877                retn
```

构造函数

```
.text:00454710                push    ebp
.text:00454711                mov     ebp, esp
.text:00454713                sub     esp, 0CCh
.text:00454719                push    ebx
.text:0045471A                push    esi
.text:0045471B                push    edi
.text:0045471C                mov     [ebp+var_8], ecx ; 存储this指针
.text:0045471F                mov     eax, [ebp+var_8] ; eaxt保存this指针，这个地址将会作为指针保存虚函数表的首地址中
.text:00454722                mov     dword ptr [eax], offset ??_7Person@@6B@ ; const Person::`vftable'
.text:00454728                mov     eax, [ebp+var_8] ; 返回对象首地址
.text:0045472B                pop     edi
.text:0045472C                pop     esi
.text:0045472D                pop     ebx
.text:0045472E                mov     esp, ebp
.text:00454730                pop     ebp
.text:00454731                retn
.text:00454731 sub_454710     endp
.text:00454731
```

11.1.2.调用自身类的虚函数

直接通过对象调用自身的成员虚函数，编译器使用了直接调用函数方式，没有访问虚表指针，而是间接获取虚函数地址。

debug

```
#include <stdio.h>

class Person {
public:
    virtual int getAge() { //虚函数定义
        return age;
    }
    virtual void setAge(int age) { //虚函数定义
        this->age = age;
    }
private:
```

```
    int age;
};

int main(int argc, char* argv[]) {
    Person person;
    person.setAge(20);
    printf("%d\n", person.getAge());
    return 0;
}
```

ida

```
.text:004548E0                push    ebp
.text:004548E1                mov     ebp, esp
.text:004548E3                sub     esp, 0D4h
.text:004548E9                push    ebx
.text:004548EA                push    esi
.text:004548EB                push    edi
.text:004548EC                lea     edi, [ebp+var_14]
.text:004548EF                mov     ecx, 5
.text:004548F4                mov     eax, 0CCCCCCCCh
.text:004548F9                rep stosd
.text:004548FB                mov     eax, ___security_cookie
.text:00454900                xor     eax, ebp
.text:00454902                mov     [ebp+var_4], eax
.text:00454905                mov     ecx, offset unk_51C003
.text:0045490A                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045490F                lea     ecx, [ebp+var_10] ; 传递this指针
.text:00454912                call    sub_451025      ; 调用默认构造函数
.text:00454917                push    14h
.text:00454919                lea     ecx, [ebp+var_10] ; 传递this指针
.text:0045491C                call    sub_450CBA      ; 直接调用函数setAge
.text:00454921                lea     ecx, [ebp+var_10] ; 传递this指针
.text:00454924                call    sub_44F9EB      ; 直接调用函数getAge
.text:00454929                push    eax
.text:0045492A                push    offset unk_4F2E60
.text:0045492F                call    sub_44EA4B
.text:00454934                add     esp, 8
.text:00454937                xor     eax, eax
.text:00454939                push    edx
.text:0045493A                mov     ecx, ebp        ; Esp
.text:0045493C                push    eax
.text:0045493D                lea     edx, Fd         ; Fd
.text:00454943                call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
```

setAge函数

```
.text:004547B0                push    ebp
.text:004547B1                mov     ebp, esp
.text:004547B3                sub     esp, 0CCh
.text:004547B9                push    ebx
.text:004547BA                push    esi
.text:004547BB                push    edi
.text:004547BC                push    ecx
.text:004547BD                lea     edi, [ebp+var_C]
.text:004547C0                mov     ecx, 3
.text:004547C5                mov     eax, 0CCCCCCCCh
.text:004547CA                rep stosd
.text:004547CC                pop     ecx
.text:004547CD                mov     [ebp+var_8], ecx
.text:004547D0                mov     ecx, offset unk_51C003
.text:004547D5                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:004547DA                mov     eax, [ebp+var_8] ; eax = this
.text:004547DD                mov     ecx, [ebp+arg_0] ; age
.text:004547E0                mov     [eax+4], ecx    ; this->age = age
.text:004547E3                pop     edi
.text:004547E4                pop     esi
.text:004547E5                pop     ebx
.text:004547E6                add     esp, 0CCh
.text:004547EC                cmp     ebp, esp
.text:004547EE                call    j___RTC_CheckEsp
.text:004547F3                mov     esp, ebp
.text:004547F5                pop     ebp
.text:004547F6                retn    4
.text:004547F6 sub_4547B0     endp
```

11.1.3.析构函数分析

执行析构函数时，实际上是在还原虚表指针，让其指向自身的虚表首地址，防止在析构函数中调用虚函数时取到非自身虚表，从而导致函数调用错误。

debug

```
#include <stdio.h>

class Person {
public:
    ~Person() {
        printf("~Person()\n");
    }
public:
    virtual int getAge() { //虚函数定义
        return age;
    }
    virtual void setAge(int age) { //虚函数定义
        this->age = age;
    }
private:
    int age;
};

int main(int argc, char* argv[]) {
    Person person;
    person.setAge(20);
    printf("%d\n", person.getAge());
    return 0;
}
```

ida析构函数分析

```
.text:00454760                push    ebp
.text:00454761                mov     ebp, esp
.text:00454763                sub     esp, 0CCh
.text:00454769                push    ebx
.text:0045476A                push    esi
.text:0045476B                push    edi
.text:0045476C                push    ecx
.text:0045476D                lea     edi, [ebp+var_C]
.text:00454770                mov     ecx, 3
.text:00454775                mov     eax, 0CCCCCCCCh
.text:0045477A                rep stosd
.text:0045477C                pop     ecx
.text:0045477D                mov     [ebp+var_8], ecx ; 保存this指针
.text:00454780                mov     ecx, offset unk_51C003
.text:00454785                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045478A                mov     eax, [ebp+var_8] ; eax得到this指针，这是虚表的位置
.text:0045478D                mov     dword ptr [eax], offset ??_7Person@@6B@ ; 将当前类虚表首地址赋值到虚表指针中
.text:00454793                push    offset aPerson  ; "~Person()\n"
.text:00454798                call    sub_44EA4B
.text:0045479D                add     esp, 4
.text:004547A0                pop     edi
```

11.2.虚函数的识别

判断是否为虚函数

- 类中隐式定义了一个数据成员
- 该数据成员在首地址处，占一个指针大小
- 构造函数会将此数据成员初始化为某个数组的首地址
- 这个地址属于数据区，是相当固定的地址

- 在这个数组中，每个元素都是函数地址
- 这些函数被调用时，第一个参数是this指针
- 在这些函数内部，很有可能堆this指针使用间接的访问方式

十二、从内存角度看继承和多重继承

12.1.识别类与类之间的关系

debug

```
#include <stdio.h>

class Base { //基类定义
public:
    Base() {
        printf("Base\n");
    }
    ~Base() {
        printf("~Base\n");
    }
    void setNumber(int n) {
        base = n;
    }
    int getNumber() {
        return base;
    }
public:
    int base;
};

class Derive : public Base { //派生类定义
public:
    void showNumber(int n) {
        setNumber(n);
        derive = n + 1;
        printf("%d\n", getNumber());
        printf("%d\n", derive);
    }
public:
    int derive;
};

int main(int argc, char* argv[]) {
    Derive derive;
    derive.showNumber(argc);
    return 0;
}
```

ida

```
.text:00454B29                mov     eax, ___security_cookie
.text:00454B2E                xor     eax, ebp
.text:00454B30                mov     [ebp+var_10], eax
.text:00454B33                push    eax
.text:00454B34                lea     eax, [ebp+var_C]
.text:00454B37                mov     large fs:0, eax
.text:00454B3D                mov     ecx, offset unk_51D003
.text:00454B42                call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:00454B47                lea     ecx, [ebp+var_1C] ; 获取对象首地址作为this指针
.text:00454B4A                call    sub_44F63A      ; 调用Derive的构造函数
.text:00454B4F                mov     [ebp+var_4], 0
.text:00454B56                mov     eax, [ebp+8]    ; 参数argc
.text:00454B59                push    eax
.text:00454B5A                lea     ecx, [ebp+var_1C] ; 传入this指针
.text:00454B5D                call    sub_4506B6      ; 调用成员函数showNumber
.text:00454B62                mov     [ebp+var_E8], 0
.text:00454B6C                mov     [ebp+var_4], 0FFFFFFFFh
.text:00454B73                lea     ecx, [ebp+var_1C] ; 传入this指针
.text:00454B76                call    sub_44E2C6      ; 调用类Derive析构函数
.text:00454B7B                mov     eax, [ebp+var_E8]
.text:00454B81                push    edx
.text:00454B82                mov     ecx, ebp        ; Esp
.text:00454B84                push    eax
.text:00454B85                lea     edx, Fd         ; Fd
.text:00454B8B                call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
.text:00454B90                pop     eax
.text:00454B91                pop     edx
.text:00454B92                mov     ecx, [ebp+var_C]
.text:00454B95                mov     large fs:0, ecx
.text:00454B9C                pop     ecx
.text:00454B9D                pop     edi
.text:00454B9E                pop     esi
```

子类Derive构造函数

```
.text:004547C0                push    ebp
.text:004547C1                mov     ebp, esp
.text:004547C3                sub     esp, 0CCh
.text:004547C9                push    ebx
.text:004547CA                push    esi
.text:004547CB                push    edi
.text:004547CC                push    ecx
.text:004547CD                lea     edi, [ebp+var_C]
.text:004547D0                mov     ecx, 3
.text:004547D5                mov     eax, 0CCCCCCCCh
.text:004547DA                rep stosd
.text:004547DC                pop     ecx
.text:004547DD                mov     [ebp+var_8], ecx
.text:004547E0                mov     ecx, [ebp+var_8] ; 以子类对象首地址作为父类的this指针
.text:004547E3                call    sub_44FF22      ; 调用父类构造函数
.text:004547E8                mov     eax, [ebp+var_8]
.text:004547EB                pop     edi
.text:004547EC                pop     esi
.text:004547ED                pop     ebx
.text:004547EE                add     esp, 0CCh
.text:004547F4                cmp     ebp, esp
.text:004547F6                call    j___RTC_CheckEsp
.text:004547FB                mov     esp, ebp
.text:004547FD                pop     ebp
.text:004547FE                retn
.text:004547FE sub_4547C0     endp
```

子类Derive析构函数

```
.text:00454870                 push    ebp
.text:00454871                 mov     ebp, esp
.text:00454873                 sub     esp, 0CCh
.text:00454879                 push    ebx
.text:0045487A                 push    esi
.text:0045487B                 push    edi
.text:0045487C                 push    ecx
.text:0045487D                 lea     edi, [ebp+var_C]
.text:00454880                 mov     ecx, 3
.text:00454885                 mov     eax, 0CCCCCCCCh
.text:0045488A                 rep stosd
.text:0045488C                 pop     ecx
.text:0045488D                 mov     [ebp+var_8], ecx
.text:00454890                 mov     ecx, [ebp+var_8] ; 以子类对象首地址作为父类的this指针
.text:00454893                 call    sub_44F3DD       ; 调用父类析构函数
.text:00454898                 pop     edi
.text:00454899                 pop     esi
.text:0045489A                 pop     ebx
.text:0045489B                 add     esp, 0CCh
.text:004548A1                 cmp     ebp, esp
.text:004548A3                 call    j___RTC_CheckEsp
.text:004548A8                 mov     esp, ebp
.text:004548AA                 pop     ebp
.text:004548AB                 retn
.text:004548AB sub_454870      endp
```

人类说话方法的多态模拟类结构

debug

```
#include <stdio.h>

class Person { // 基类--“人”类
public:
    Person() {}
    virtual ~Person() {}
    virtual void showSpeak() {} // 这里用纯虚函数更好，相关的知识点后面会讲到
};

class Chinese : public Person { // 中国人：继承自人类
public:
    Chinese() {}
    virtual ~Chinese() {}
    virtual void showSpeak() { // 覆盖基类虚函数
        printf("Speak Chinese\r\n");
    }
};

class American : public Person { //美国人：继承自人类
public:
    American() {}
    virtual ~American() {}
    virtual void showSpeak() { //覆盖基类虚函数
        printf("Speak American\r\n");
    }
};

class German : public Person { //德国人：继承自人类
public:
    German() {}
    virtual ~German() {}
    virtual void showSpeak() { //覆盖基类虚函数
```

```
        printf("Speak German\r\n");
    }
};

void speak(Person* person) { //根据虚表信息获取虚函数首地址并调用
    person->showSpeak();
}

int main(int argc, char* argv[]) {
    Chinese chinese;
    American american;
    German german;
    speak(&chinese);
    speak(&american);
    speak(&german);
    return 0;
}
```

speak函数分析，虚函数的调用过程是间接寻址方式

```
.text:00454E77 __$EncStackInitEnd_15:                  ; JMC_flag
.text:00454E77                 mov     ecx, offset _8E097BDB_main@cpp
.text:00454E7C                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:00454E81                 mov     eax, [ebp+person] ; eax获取参数person的值
.text:00454E84                 mov     edx, [eax]      ; 取虚表首地址并传递给edx
.text:00454E86                 mov     esi, esp
.text:00454E88                 mov     ecx, [ebp+person] ; 传递this指针
.text:00454E8B                 mov     eax, [edx+4]    ; 利用虚表指针edx，间接调用函数
.text:00454E8E                 call    eax
.text:00454E90                 cmp     esi, esp
.text:00454E92                 call    j___RTC_CheckEsp
```

12.2.多重继承

debug

```
#include <stdio.h>

class Sofa {
public:
    Sofa() {
        color = 2;
    }
    virtual ~Sofa() { // 沙发类虚析构函数
        printf("virtual ~Sofa()\n");
    }
    virtual int getColor() { // 获取沙发颜色
        return color;
    }
    virtual int sitDown() { // 沙发可以坐下休息
        return printf("Sit down and rest your legs\r\n");
    }
protected:
    int color; // 沙发类成员变量
};

//定义床类
class Bed {
public:
```

```
    Bed() {
        length = 4;
        width = 5;
    }
    virtual ~Bed() { //床类虚析构函数
        printf("virtual ~Bed()\n");
    }
    virtual int getArea() { //获取床面积
        return length * width;
    }
    virtual int sleep() { //床可以用来睡觉
        return printf("go to sleep\r\n");
    }
protected:
    int length; //床类成员变量
    int width;
};

//子类沙发床定义，派生自Sofa类和Bed类
class SofaBed : public Sofa, public Bed {
public:
    SofaBed() {
        height = 6;
    }
    virtual ~SofaBed() { //沙发床类的虚析构函数
        printf("virtual ~SofaBed()\n");
    }
    virtual int sitDown() { //沙发可以坐下休息
        return printf("Sit down on the sofa bed\r\n");
    }
    virtual int sleep() { //床可以用来睡觉
        return printf("go to sleep on the sofa bed\r\n");
    }
    virtual int getHeight() {
        return height;
    }
protected:
    int height;
};

int main(int argc, char* argv[]) {
    SofaBed sofabed;
    return 0;
}
```

ida

```
.text:00454FE0
.text:00454FE0                 push    ebp
.text:00454FE1                 mov     ebp, esp
.text:00454FE3                 sub     esp, 0F0h
.text:00454FE9                 push    ebx
.text:00454FEA                 push    esi
.text:00454FEB                 push    edi
.text:00454FEC                 lea     edi, [ebp+var_30]
.text:00454FEF                 mov     ecx, 0Ch
.text:00454FF4                 mov     eax, 0CCCCCCCCh
.text:00454FF9                 rep stosd
.text:00454FFB                 mov     eax, ___security_cookie
.text:00455000                 xor     eax, ebp
.text:00455002                 mov     [ebp+var_4], eax
.text:00455005                 mov     ecx, offset unk_51D003
.text:0045500A                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045500F                 lea     ecx, [ebp+var_20] ; 传递this指针
.text:00455012                 call    sub_44FBB7      ; 调用构造函数
.text:00455017                 mov     [ebp+var_EC], 0
.text:00455021                 lea     ecx, [ebp+var_20]
.text:00455024                 call    sub_450120      ; 调用析构函数
.text:00455029                 mov     eax, [ebp+var_EC]
.text:0045502F                 push    edx
.text:00455030                 mov     ecx, ebp        ; Esp
.text:00455032                 push    eax
.text:00455033                 lea     edx, Fd         ; Fd
.text:00455039                 call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
.text:0045503E                 pop     eax
```

构造函数

```
.text:004548EA                 pop     ecx
.text:004548EB                 mov     eax, ___security_cookie
.text:004548F0                 xor     eax, ebp
.text:004548F2                 push    eax
.text:004548F3                 lea     eax, [ebp+var_C]
.text:004548F6                 mov     large fs:0, eax
.text:004548FC                 mov     [ebp+var_14], ecx
.text:004548FF                 mov     ecx, offset unk_51D003
.text:00454904                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:00454909                 mov     ecx, [ebp+var_14] ; 以对象首地址作为this指针
.text:0045490C                 call    sub_4512AA      ; 调用沙发父类的构造函数
.text:00454911                 mov     [ebp+var_4], 0
.text:00454918                 mov     ecx, [ebp+var_14]
.text:0045491B                 add     ecx, 8          ; 将this指针调整到第二个虚表指针的地址处
.text:0045491E                 call    sub_44F6B2      ; 调用床父类的构造函数
.text:00454923                 mov     eax, [ebp+var_14] ; 获取对象的首地址
.text:00454926                 mov     dword ptr [eax], offset ??_7SofaBed@@6B@ ; const SofaBed::`vftable'
.text:0045492C                 mov     eax, [ebp+var_14]
.text:0045492F                 mov     dword ptr [eax+8], offset ??_7SofaBed@@6B@_0 ; const SofaBed::`vftable'
.text:00454936                 mov     eax, [ebp+var_14]
.text:00454939                 mov     dword ptr [eax+14h], 6
.text:00454940                 mov     [ebp+var_4], 0FFFFFFFFh
.text:00454947                 mov     eax, [ebp+var_14]
.text:0045494A                 mov     ecx, [ebp+var_C]
.text:0045494D                 mov     large fs:0, ecx
.text:00454954                 pop     ecx
.text:00454955                 pop     edi
.text:00454956                 pop     esi
.text:00454957                 pop     ebx
```

析构函数

```
.text:00454A80                 push    ebp
.text:00454A81                 mov     ebp, esp
.text:00454A83                 sub     esp, 0CCh
.text:00454A89                 push    ebx
.text:00454A8A                 push    esi
.text:00454A8B                 push    edi
.text:00454A8C                 push    ecx
.text:00454A8D                 lea     edi, [ebp+var_C]
.text:00454A90                 mov     ecx, 3
.text:00454A95                 mov     eax, 0CCCCCCCCh
.text:00454A9A                 rep stosd
.text:00454A9C                 pop     ecx
.text:00454A9D                 mov     [ebp+var_8], ecx
.text:00454AA0                 mov     ecx, offset unk_51D003
.text:00454AA5                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:00454AAA                 mov     eax, [ebp+var_8]
.text:00454AAD                 mov     dword ptr [eax], offset ??_7SofaBed@@6B@ ; const SofaBed::`vftable'
.text:00454AB3                 mov     eax, [ebp+var_8]
.text:00454AB6                 mov     dword ptr [eax+8], offset ??_7SofaBed@@6B@_0 ; const SofaBed::`vftable'
.text:00454ABD                 push    offset aVirtualSofabed ; "virtual ~SofaBed()\n"
.text:00454AC2                 call    sub_44EA73
.text:00454AC7                 add     esp, 4
.text:00454ACA                 mov     ecx, [ebp+var_8]
.text:00454ACD                 add     ecx, 8
.text:00454AD0                 call    sub_44F973
.text:00454AD5                 mov     ecx, [ebp+var_8]
.text:00454AD8                 call    sub 450459
```

单继承类和多重继承类特征总结

单继承

- 在类对象占用的内存空间中，只保存一份虚表指针。
- 因为只有一个虚表指针，所以只有一个虚表。
- 虚表中各项保存了类中各虚函数的首地址。
- 构造时先构造父类，再构造自身，并且只调用一次父类构造函数。
- 析构时先析构自身，再析构父类，并且只调用一次父类析构函数

多重继承

- 在类对象占用内存空间中，根据继承父类（有虚函数）个数保存对应的虚表指针。
- 根据保存的虚表指针的个数，产生相应个数的虚表。
- 转换父类指针时，需要调整到对象的首地址。
- 构造时需要调用多个父类构造函数。
- 构造时先构造继承列表中的第一个父类，然后依次调用到最后一个继承的父类构造函数。
- 析构时先析构自身，然后以构造函数相反的顺序调用所有父类的析构函数。
- 当对象作为成员时，整个类对象的内存结构和多重继承相似。当类中无虚函数时，整个类对象内存结构和多重继承完全一样，可酌情还原。当父类或成员对象存在虚函数时，通过观察虚表指针的位置和构造、析构函数中填写虚表指针的数目、顺序及目标地址，还原继承或成员关系。

12.3.抽象类

debug

```
#include <stdio.h>

class AbstractBase {
public:
    AbstractBase() {
        printf("AbstractBase()");
    }
    virtual void show() = 0; //定义纯虚函数
};

class VirtualChild : public AbstractBase { //定义继承抽象类的子类
public:
    virtual void show() { //实现纯虚函数
        printf("抽象类分析\n");
    }
};

int main(int argc, char* argv[]) {
    VirtualChild obj;
    obj.show();
    return 0;
}
```

ida

```
.text:0045493B __$EncStackInitEnd_4:
.text:0045493B                 mov     eax, ___security_cookie
.text:00454940                 xor     eax, ebp
.text:00454942                 mov     [ebp+var_4], eax
.text:00454945                 mov     ecx, offset _8E097BDB_main@cpp ; JMC_flag
.text:0045494A                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045494F                 lea     ecx, [ebp+obj]  ; this
.text:00454952                 call    j_??0VirtualChild@@QAE@XZ ; VirtualChild::VirtualChild(void)
.text:00454957                 lea     ecx, [ebp+obj]  ; this
.text:0045495A                 call    j_?show@VirtualChild@@UAEXXZ ; VirtualChild::show(void)
.text:0045495F                 xor     eax, eax
.text:00454961                 push    edx
.text:00454962                 mov     ecx, ebp        ; frame
.text:00454964                 push    eax
.text:00454965                 lea     edx, v          ; v
.text:0045496B                 call    j_@_RTC_CheckStackVars@8 ; _RTC_CheckStackVars(x,x)
```

子类构造函数

```
.text:004547AC __$EncStackInitEnd_0:
.text:004547AC                 pop     ecx
.text:004547AD                 mov     [ebp+this], ecx
.text:004547B0                 mov     ecx, [ebp+this] ; this
.text:004547B3                 call    j_??0AbstractBase@@QAE@XZ ; AbstractBase::AbstractBase(void)
.text:004547B8                 mov     eax, [ebp+this]
.text:004547BB                 mov     dword ptr [eax], offset ??_7VirtualChild@@6B@ ; const VirtualChild::`vftable'
.text:004547C1                 mov     eax, [ebp+this]
.text:004547C4                 pop     edi
.text:004547C5                 pop     esi
.text:004547C6                 pop     ebx
.text:004547C7                 add     esp, 0CCh
.text:004547CD                 cmp     ebp, esp
.text:004547CF                 call    j___RTC_CheckEsp
.text:004547D4                 mov     esp, ebp
.text:004547D6                 pop     ebp
.text:004547D7                 retn
.text:004547D7 ??0VirtualChild@@QAE@XZ endp
```

抽象类构造函数

```
.text:0045473C __$EncStackInitEnd:
.text:0045473C                 pop     ecx
.text:0045473D                 mov     [ebp+this], ecx
.text:00454740                 mov     ecx, offset _8E097BDB_main@cpp ; JMC_flag
.text:00454745                 call    j_@__CheckForDebuggerJustMyCode@4 ; __CheckForDebuggerJustMyCode(x)
.text:0045474A                 mov     eax, [ebp+this]
.text:0045474D                 mov     dword ptr [eax], offset ??_7AbstractBase@@6B@ ; const AbstractBase::`vftable'
.text:00454753                 push    offset _Format  ; "AbstractBase()"
.text:00454758                 call    j__printf
.text:0045475D                 add     esp, 4
.text:00454760                 mov     eax, [ebp+this]
.text:00454763                 pop     edi
.text:00454764                 pop     esi
.text:00454765                 pop     ebx
.text:00454766                 add     esp, 0CCh
.text:0045476C                 cmp     ebp, esp
.text:0045476E                 call    j___RTC_CheckEsp
.text:00454773                 mov     esp, ebp
.text:00454775                 pop     ebp
.text:00454776                 retn
.text:00454776 ??0AbstractBase@@QAE@XZ endp
```

抽象类的虚表信息

```
.rdata:004F2E54 ; void (__cdecl *const AbstractBase::`vftable'[2])()
.rdata:004F2E54 ??_7AbstractBase@@6B@ dd offset j___purecall
.rdata:004F2E54                                         ; DATA XREF: AbstractBase::AbstractBase(void)+2D↑o
```

在抽象类的虚表信息中，因为纯虚函数没有实现代码，所以没有首地址，编译器为了防止误调用虚函数，将虚表

中保存的纯虚函数的首地址项替换成函数`__purecall`，用于结束程序。在分析过程中，一旦在虚表中发现函数地址为`__purecall`函数时，就可以高度怀疑此虚表对应的类是一个抽象类。