作业题目 4：乘法指令和过程调用

一. 输出九九乘法表

基本要求：至少实现一个**过程调用**，能正确调用并返回。（Call 和 ret 指令）

结果输出：

```
C:\>9
The 9mul9 table:
9*1=9   9*2=18   9*3=27   9*4=36   9*5=45   9*6=54   9*7=63   9*8=72   9*9=81
8*1=8   8*2=16   8*3=24   8*4=32   8*5=40   8*6=48   8*7=56   8*8=64
7*1=7   7*2=14   7*3=21   7*4=28   7*5=35   7*6=42   7*7=49
6*1=6   6*2=12   6*3=18   6*4=24   6*5=30   6*6=36
5*1=5   5*2=10   5*3=15   5*4=20   5*5=25
4*1=4   4*2=8   4*3=12   4*4=16
3*1=3   3*2=6   3*3=9
2*1=2   2*2=4
1*1=1

C:\>
```

1）用 Call 和 ret 指令实现过程调用，过程（process）中需要用到主过程中寄存器资源（掌握寄存器在过程调用前后的保存和恢复）。

2）复习双循环的实现；

3）用 C 语言实现后察看反汇编代码并加注释；

结果输出：

```
C:\>mul.exe
The 9mu19 table:
9x1=9   9x2=18  9x3=27  9x4=36  9x5=45  9x6=54  9x7=63  9x8=72  9x9=81
8x1=8   8x2=16  8x3=24  8x4=32  8x5=40  8x6=48  8x7=56  8x8=64
7x1=7   7x2=14  7x3=21  7x4=28  7x5=35  7x6=42  7x7=49
6x1=6   6x2=12  6x3=18  6x4=24  6x5=30  6x6=36
5x1=5   5x2=10  5x3=15  5x4=20  5x5=25
4x1=4   4x2=8   4x3=12  4x4=16
3x1=3   3x2=6   3x3=9
2x1=2   2x2=4
1x1=1

C:\>_
```

用 C 语言实现后查看反汇编代码：

```
00000001400013bd <main>:
{
        int result = num1 * num2;
        printf("%d*%d=%d\t", num1, num2, result);
}
int main()
{
   1400013bd:   55                      push   %rbp
   1400013be:   48 89 e5                mov    %rsp,%rbp
   1400013c1:   48 83 ec 30             sub    $0x30,%rsp
   1400013c5:   e8 c2 00 00 00          call   14000148c <__main>
        for (int i = 9; i >= 1 ; i--) {
   1400013ca:   c7 45 fc 09 00 00 00    movl   $0x9,-0x4(%rbp)
   1400013d1:   eb 30                   jmp    140001403 <main+0x46>
                for (int j = 1; j <= i; j++) {
   1400013d3:   c7 45 f8 01 00 00 00    movl   $0x1,-0x8(%rbp)
   1400013da:   eb 11                   jmp    1400013ed <main+0x30>
                        mul(i, j);
   1400013dc:   8b 55 f8                mov    -0x8(%rbp),%edx
   1400013df:   8b 45 fc                mov    -0x4(%rbp),%eax
   1400013e2:   89 c1                   mov    %eax,%ecx
   1400013e4:   e8 97 ff ff ff          call   140001380 <mul>
                for (int j = 1; j <= i; j++) {
   1400013e9:   83 45 f8 01             addl   $0x1,-0x8(%rbp)
   1400013ed:   8b 45 f8                mov    -0x8(%rbp),%eax
   1400013f0:   3b 45 fc                cmp    -0x4(%rbp),%eax
   1400013f3:   7e e7                   jle    1400013dc <main+0x1f>
                }
                printf("\n");
   1400013f5:   b9 0a 00 00 00          mov    $0xa,%ecx
   1400013fa:   e8 59 46 00 00          call   140005a58 <putchar>
        for (int i = 9; i >= 1 ; i--) {
   1400013ff:   83 6d fc 01             subl   $0x1,-0x4(%rbp)
   140001403:   83 7d fc 00             cmpl   $0x0,-0x4(%rbp)
   140001407:   7f ca                   jg     1400013d3 <main+0x16>
        }
        return 0;
   140001409:   b8 00 00 00 00          mov    $0x0,%eax
   14000140e:   48 83 c4 30             add    $0x30,%rsp
   140001412:   5d                      pop    %rbp
   140001413:   c3                      ret
   140001414:   90                      nop
   140001415:   90                      nop
   140001416:   90                      nop
   140001417:   90                      nop
   140001418:   90                      nop
   140001419:   90                      nop
```

int main()

{

1400013bd: 55 push %rbp ；保存旧的基址指针

1400013be: 48 89 e5 mov %rsp,%rbp ；设置新的基址指针

1400013c1: 48 83 ec 30 sub $0x30,%rsp ；在栈上分配 48 字节空间

1400013c5: e8 c2 00 00 00 call 14000148c <__main> ；调用运行时初始化

```
    for (int i = 9; i >= 1 ; i--) {
    1400013ca: c7 45 fc 09 00 00 00 movl    $0x9,-0x4(%rbp)              ; i = 9
    1400013d1: eb 30                 jmp    140001403 <main+0x46>    ; 跳转到循环条件检查

    for (int j = 1; j <= i; j++)
    1400013d3: c7 45 f8 01 00 00 00 movl    $0x1,-0x8(%rbp)              ; j = 1
    1400013da: eb 11                 jmp    1400013ed <main+0x30>    ; 跳转到内层循环条件检查

    mul(i, j);
    1400013dc: 8b 55 f8              mov    -0x8(%rbp),%edx              ; 加载 j 到 EDX（第二个参数）
    1400013df: 8b 45 fc              mov    -0x4(%rbp),%eax              ; 加载 i 到 EAX
    1400013e2: 89 c1                 mov    %eax,%ecx                    ; 移动 i 到 ECX（第一个参数）
    1400013e4: e8 97 ff ff ff        call   140001380 <mul>              ; 调用 mul 函数

    for (int j = 1; j <= i; j++)
    1400013e9: 83 45 f8 01           addl   $0x1,-0x8(%rbp)              ; j++
    1400013ed: 8b 45 f8              mov    -0x8(%rbp),%eax              ; 加载 j 到 EAX
    1400013f0: 3b 45 fc              cmp    -0x4(%rbp),%eax              ; 比较 j 和 i
    1400013f3: 7e e7                 jle    1400013dc <main+0x1f>    ; 如果 j <= i，继续内层循环

    printf("\n");
    1400013f5: b9 0a 00 00 00        mov    $0xa,%ecx                    ; 加载换行符'\n'到 ECX
    1400013fa: e8 59 46 00 00        call   140005a58 <putchar>          ; 调用 putchar 输出换行

    for (int i = 9; i >= 1 ; i--) {
    1400013ff: 83 6d fc 01           subl   $0x1,-0x4(%rbp)              ; i--
    140001403: 83 7d fc 00           cmpl   $0x0,-0x4(%rbp)              ; 比较 i 和 0
    140001407: 7f ca                 jg     1400013d3 <main+0x16>    ; 如果 i > 0，继续外层循环

    return 0;
    140001409: b8 00 00 00 00        mov    $0x0,%eax                    ; 返回值 0
    14000140e: 48 83 c4 30           add    $0x30,%rsp                   ; 释放栈空间
    140001412: 5d                    pop    %rbp                         ; 恢复旧的基址指针
    140001413: c3                    ret                                 ; 返回操作系统
```

140001414: 90 nop ; 空操作

140001415: 90 nop ; 空操作

140001416: 90 nop ; 空操作

}

```
void mul(int num1, int num2)
{
   140001380:   55                      push   %rbp
   140001381:   48 89 e5                mov    %rsp,%rbp
   140001384:   48 83 ec 30             sub    $0x30,%rsp
   140001388:   89 4d 10                mov    %ecx,0x10(%rbp)
   14000138b:   89 55 18                mov    %edx,0x18(%rbp)
       int result = num1 * num2;
   14000138e:   8b 45 10                mov    0x10(%rbp),%eax
   140001391:   0f af 45 18             imul   0x18(%rbp),%eax
   140001395:   89 45 fc                mov    %eax,-0x4(%rbp)
       printf("%d*%d=%d\t", num1, num2, result);
   140001398:   44 8b 45 fc             mov    -0x4(%rbp),%r8d
   14000139c:   8b 55 18                mov    0x18(%rbp),%edx
   14000139f:   8b 45 10                mov    0x10(%rbp),%eax
   1400013a2:   48 8d 0d 57 5c 00 00    lea    0x5c57(%rip),%rcx        # 140007000 <.rdata>
   1400013a9:   45 89 c1                mov    %r8d,%r9d
   1400013ac:   41 89 d0                mov    %edx,%r8d
   1400013af:   89 c2                   mov    %eax,%edx
   1400013b1:   e8 7a 0c 00 00          call   140002030 <__mingw_printf>
}
   1400013b6:   90                      nop
   1400013b7:   48 83 c4 30             add    $0x30,%rsp
   1400013bb:   5d                      pop    %rbp
   1400013bc:   c3                      ret
```

int main()

{

1400013bd: 55 push %rbp ; 保存旧的基址指针

1400013be: 48 89 e5 mov %rsp,%rbp ; 设置新的基址指针

1400013c1: 48 83 ec 30 sub $0x30,%rsp ; 在栈上分配 48 字节空间

1400013c5: e8 c2 00 00 00 call 14000148c <__main> ; 调用运行时初始化

for (int i = 9; i >= 1 ; i--) {

1400013ca: c7 45 fc 09 00 00 00 movl $0x9,-0x4(%rbp) ; i = 9

1400013d1: eb 30 jmp 140001403 <main+0x46> ; 跳转到循环条件检查

for (int j = 1; j <= i; j++)

1400013d3: c7 45 f8 01 00 00 00 movl $0x1,-0x8(%rbp) ; j = 1

1400013da: eb 11 jmp 1400013ed <main+0x30> ; 跳转到内层循环条件检查

mul(i, j);

1400013dc: 8b 55 f8 mov -0x8(%rbp),%edx ; 加载 j 到 EDX（第二个参数）

1400013df: 8b 45 fc mov -0x4(%rbp),%eax ; 加载 i 到 EAX

1400013e2: 89 c1 mov %eax,%ecx ; 移动 i 到 ECX

（第一个参数）

```
    1400013e4: e8 97 ff ff ff          call    140001380 <mul>             ; 调用 mul 函数

     for (int j = 1; j <= i; j++)
    1400013e9: 83 45 f8 01              addl    $0x1,-0x8(%rbp)             ; j++
    1400013ed: 8b 45 f8                 mov     -0x8(%rbp),%eax             ; 加载 j 到 EAX
    1400013f0: 3b 45 fc                 cmp     -0x4(%rbp),%eax             ; 比较 j 和 i
    1400013f3: 7e e7                    jle     1400013dc <main+0x1f>       ; 如果 j <= i，继续内
层循环

     printf("\n");
    1400013f5: b9 0a 00 00 00           mov     $0xa,%ecx                   ; 加载换行符'\n'到
ECX
    1400013fa: e8 59 46 00 00           call    140005a58 <putchar>         ; 调用 putchar 输出换
行

     for (int i = 9; i >= 1 ; i--) {
    1400013ff: 83 6d fc 01              subl    $0x1,-0x4(%rbp)             ; i--
    140001403: 83 7d fc 00              cmpl    $0x0,-0x4(%rbp)             ; 比较 i 和 0
    140001407: 7f ca                    jg      1400013d3 <main+0x16>       ; 如果 i > 0，继续
外层循环

     return 0;
    140001409: b8 00 00 00 00           mov     $0x0,%eax                   ; 返回值 0
    14000140e: 48 83 c4 30              add     $0x30,%rsp                  ; 释放栈空间
    140001412: 5d                       pop     %rbp                        ; 恢复旧的基址
指针
    140001413: c3                       ret                                 ; 返回操作系统
    140001414: 90                       nop                                 ; 空操作
    140001415: 90                       nop                                 ; 空操作
    140001416: 90                       nop                                 ; 空操作
}
```

作业题目 4：乘法指令和过程调用

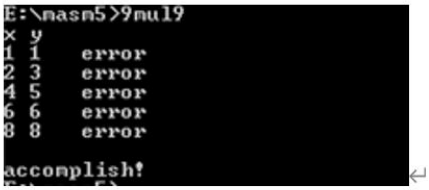
二· 九九乘法表纠错

基本要求：检查 9*9 乘法表内数据是否正确，将不正确位置确定下来并显示在屏幕上；

如数据部分：

```
data segment
table  db 7,2,3,4,5,6,7,8,9 ;9*9 表数据
  → db 2,4,7,8,10,12,14,16,18
  → db 3,6,9,12,15,18,21,24,27
  → db 4,8,12,16,7,24,28,32,36
  → db 5,10,15,20,25,30,35,40,45
  → db 6,12,18,24,30,7,42,48,54
  → db 7,14,21,28,35,42,49,56,63
  → db 8,16,24,32,40,48,56,7,72
  → db 9,18,27,36,45,54,63,72,81
……
data ends
```

检查结果：



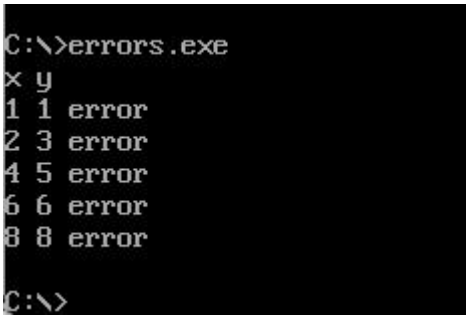
1）用 Call 和 ret 指令实现过程调用，过程（process）中需要用到主过程中寄存器资源（掌握寄存器在过程调用前后的保存和恢复）。

2）复习双循环的实现；

3）注意数据段的寻址方式；

4）用 C 语言实现后察看反汇编代码并加注释；
观察把数据分别定义为全局和局部的区别。

结果输出：



C 语言反汇编结果以及备注如图所示：

```
int main()
{
   1400013ec:   55                      push   %rbp
   1400013ed:   48 89 e5                mov    %rsp,%rbp
   1400013f0:   48 83 ec 30             sub    $0x30,%rsp
   1400013f4:   e8 c3 00 00 00          call   1400014bc <__main>
       printf("x y\n");
   1400013f9:   48 8d 05 0d 5c 00 00    lea    0x5c0d(%rip),%rax        # 14000700d <.rdata+0xd>
   140001400:   48 89 c1                mov    %rax,%rcx
   140001403:   e8 80 46 00 00          call   140005a88 <puts>
       for (int i = 1; i <= 9 ; i++) {
   140001408:   c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)
   14000140f:   eb 24                   jmp    140001435 <main+0x49>
               for (int j = 1; j <=9; j++) {
   140001411:   c7 45 f8 01 00 00 00    movl   $0x1,-0x8(%rbp)
   140001418:   eb 11                   jmp    14000142b <main+0x3f>
                       mul(i, j);
   14000141a:   8b 55 f8                mov    -0x8(%rbp),%edx
   14000141d:   8b 45 fc                mov    -0x4(%rbp),%eax
   140001420:   89 c1                   mov    %eax,%ecx
   140001422:   e8 59 ff ff ff          call   140001380 <mul>
               for (int j = 1; j <=9; j++) {
   140001427:   83 45 f8 01             addl   $0x1,-0x8(%rbp)
   14000142b:   83 7d f8 09             cmpl   $0x9,-0x8(%rbp)
   14000142f:   7e e9                   jle    14000141a <main+0x2e>
       for (int i = 1; i <= 9 ; i++) {
   140001431:   83 45 fc 01             addl   $0x1,-0x4(%rbp)
   140001435:   83 7d fc 09             cmpl   $0x9,-0x4(%rbp)
   140001439:   7e d6                   jle    140001411 <main+0x25>
               }
       }
       return 0;
   14000143b:   b8 00 00 00 00          mov    $0x0,%eax
   140001440:   48 83 c4 30             add    $0x30,%rsp
   140001444:   5d                      pop    %rbp
   140001445:   c3                      ret
   140001446:   90                      nop
```

int main()

{

1400013ec: 55 push %rbp ；保存旧的栈帧基址

1400013ed: 48 89 e5 mov %rsp,%rbp ；设置新的栈帧基址

1400013f0: 48 83 ec 30 sub $0x30,%rsp ；为局部变量分配 48 字节栈空间

1400013f4: e8 c3 00 00 00 call 1400014bc <__main> ；调用 GCC 初始化函数

printf("x y\n");

1400013f9: 48 8d 05 04 50 00 00 lea 0x5004(%rip),%rax # 14000700d <.rdata+0xd> ；加载字符串地址

140001400: 48 89 c1 mov %rax,%rcx ；第一个参数放入 rcx (Windows 调用约定)

140001403: e8 80 46 00 00 call 140005a88 <puts> ；调用 puts 输出字符串

for (int i = 1; i <= 9 ; i++) {

140001408: c7 45 fc 01 00 00 00 movl $0x1,-0x4(%rbp) ; i = 1 存储在栈[rbp-4]

14000140f: eb 24 jmp 140001435 <main+0x49> ；跳转到循环条件检查

for (int j = 1; j <=9; j++) {

140001411: c7 45 f8 01 00 00 00 movl $0x1,-0x8(%rbp) ; j = 1 存储在栈[rbp-8]

140001418: eb 11 jmp 14000142b <main+0x3f> ；跳转到内层循环条件检查

```
                mul(i, j);
    14000141a: 8b 55 f8                 mov     -0x8(%rbp),%edx     ; j 放入 edx (第二个参
数)
    14000141d: 8b 45 fc                 mov     -0x4(%rbp),%eax     ; i 放入 eax
    140001420: 89 c1                    mov     %eax,%ecx           ; i 放入 ecx (第一个参
数)
    140001422: e8 59 ff ff ff           call    140001380 <mul>     ; 调用 mul 函数

        for (int j = 1; j <=9; j++) {
    140001427: 83 45 f8 01              addl    $0x1,-0x8(%rbp)     ; j++
    14000142b: 83 7d f8 09              cmpl    $0x9,-0x8(%rbp)     ; 比较 j <= 9
    14000142f: 7e e9                    jle     14000141a <main+0x2e> ; 如果 j<=9，继续内
层循环

     for (int i = 1; i <= 9 ; i++) {
    140001431: 83 45 fc 01              addl    $0x1,-0x4(%rbp)     ; i++
    140001435: 83 7d fc 09              cmpl    $0x9,-0x4(%rbp)     ; 比较 i <= 9
    140001439: 7e d6                    jle     140001411 <main+0x25> ; 如果 i<=9，继续外
层循环
        }
     }
     return 0;
    14000143b: b8 00 00 00 00           mov     $0x0,%eax           ; 返回值 0
    140001440: 48 83 c4 30              add     $0x30,%rsp          ; 恢复栈指针
    140001444: 5d                       pop     %rbp                ; 恢复旧的栈帧基址
    140001445: c3                       ret                         ; 返回
    140001446: 90                       nop                         ; 对齐填充
```

```
void mul(int num1, int num2)
{
   140001380:   55                      push   %rbp
   140001381:   48 89 e5                mov    %rsp,%rbp
   140001384:   48 83 ec 30             sub    $0x30,%rsp
   140001388:   89 4d 10                mov    %ecx,0x10(%rbp)
   14000138b:   89 55 18                mov    %edx,0x18(%rbp)
       int result = num1 * num2;
   14000138e:   8b 45 10                mov    0x10(%rbp),%eax
   140001391:   0f af 45 18             imul   0x18(%rbp),%eax
   140001395:   89 45 fc                mov    %eax,-0x4(%rbp)
       if (result != a[num1-1][num2-1]) {
   140001398:   8b 45 10                mov    0x10(%rbp),%eax
   14000139b:   8d 50 ff                lea    -0x1(%rax),%edx
   14000139e:   8b 45 18                mov    0x18(%rbp),%eax
   1400013a1:   83 e8 01                sub    $0x1,%eax
   1400013a4:   48 63 c8                movslq %eax,%rcx
   1400013a7:   48 63 d2                movslq %edx,%rdx
   1400013aa:   48 89 d0                mov    %rdx,%rax
   1400013ad:   48 c1 e0 03             shl    $0x3,%rax
   1400013b1:   48 01 d0                add    %rdx,%rax
   1400013b4:   48 01 c8                add    %rcx,%rax
   1400013b7:   48 8d 14 85 00 00 00    lea    0x0(,%rax,4),%rdx
   1400013be:   00
   1400013bf:   48 8d 05 3a 4c 00 00    lea    0x4c3a(%rip),%rax        # 140006000 <__data_start__>
   1400013c6:   8b 04 02                mov    (%rdx,%rax,1),%eax
   1400013c9:   39 45 fc                cmp    %eax,-0x4(%rbp)
   1400013cc:   74 17                   je     1400013e5 <mul+0x65>
               printf("%d %d:error\n", num1, num2);
   1400013ce:   8b 55 18                mov    0x18(%rbp),%edx
   1400013d1:   8b 45 10                mov    0x10(%rbp),%eax
   1400013d4:   48 8d 0d 25 5c 00 00    lea    0x5c25(%rip),%rcx        # 140007000 <.rdata>
   1400013db:   41 89 d0                mov    %edx,%r8d
   1400013de:   89 c2                   mov    %eax,%edx
   1400013e0:   e8 7b 0c 00 00          call   140002060 <__mingw_printf>
       }
}
   1400013e5:   90                      nop
   1400013e6:   48 83 c4 30             add    $0x30,%rsp
   1400013ea:   5d                      pop    %rbp
   1400013eb:   c3                      ret
```

void mul(int num1, int num2)

{

140001380: 55 push %rbp ; 保存旧的栈帧基址

140001381: 48 89 e5 mov %rsp,%rbp ; 设置新的栈帧基址

140001384: 48 83 ec 30 sub $0x30,%rsp ; 分配 48 字节栈空间

140001388: 89 4d 10 mov %ecx,0x10(%rbp) ; 保存第一个参数 num1 到[rbp+16]

14000138b: 89 55 18 mov %edx,0x18(%rbp) ; 保存第二个参数 num2 到[rbp+24]

int result = num1 * num2;

14000138e: 8b 45 10 mov 0x10(%rbp),%eax ; 加载 num1 到 eax

140001391: 0f af 45 18 imul 0x18(%rbp),%eax ; eax = num1 * num2

140001395: 89 45 fc mov %eax,-0x4(%rbp) ; 结果存储到[rbp-4]

if (result != a[num1-1][num2-1]) {

140001398: 8b 45 10 mov 0x10(%rbp),%eax ; 加载 num1

14000139b: 8d 50 ff lea -0x1(%rax),%edx ; edx = num1 - 1 (行索引)

14000139e: 8b 45 18 mov 0x18(%rbp),%eax ; 加载 num2

1400013a1: 83 e8 01 sub $0x1,%eax ; eax = num2 - 1 (列索引)

; 计算二维数组 a[num1-1][num2-1]的地址

```
    1400013a4: 48 63 c8                 movslq %eax,%rcx            ; 零扩展列索引到 64 位
    1400013a7: 48 63 d2                 movslq %edx,%rdx            ; 零扩展行索引到 64 位
    1400013aa: 48 89 d0                 mov    %rdx,%rax            ; rax =  行索引
    1400013ad: 48 c1 e0 03              shl    $0x3,%rax            ; rax *= 8 (每行 8 字节？这
里应该是 9 列，可能优化)
    1400013b1: 48 01 d0                 add    %rdx,%rax            ; rax =  行索引  * 9
    1400013b4: 48 01 c8                 add    %rcx,%rax            ; rax +=  列索引
    1400013b7: 48 8d 14 85 00 00 00     lea    0x0(,%rax,4),%rdx ; rdx = rax * 4 (每个元素 4 字
节)
    1400013be: 00
    1400013bf: 48 8d 05 3a 4c 00 00     lea    0x4c3a(%rip),%rax # 140006000 <a> ; 加载数组 a
的基地址
    1400013c6: 8b 04 02                 mov    (%rdx,%rax,1),%eax ; 加载 a[num1-1][num2-1]
的值

    1400013c9: 39 45 fc                 cmp    %eax,-0x4(%rbp)     ; 比较 result 和数组值
    1400013cc: 74 17                    je     1400013e5 <mul+0x65> ; 如果相等，跳转到
结尾

        printf("%d %d:error\n", num1, num2);
    1400013ce: 8b 55 18                 mov    0x18(%rbp),%edx     ; 加载 num2
    1400013d1: 8b 45 10                 mov    0x10(%rbp),%eax     ; 加载 num1
    1400013d4: 48 8d 0d 25 5c 00 00     lea    0x5c25(%rip),%rcx # 140007000 <.rdata> ; 加载
格式字符串
    1400013db: 41 89 d0                 mov    %edx,%r8d           ; num2 放入 r8d (第三
个参数)
    1400013de: 89 c2                    mov    %eax,%edx           ; num1 放入 edx (第二
个参数)
    1400013e0: e8 7b 0c 00 00           call   140002060 <_mingw_printf> ; 调用 printf
    }

    1400013e5: 90                       nop                        ; 对齐填充
    1400013e6: 48 83 c4 30              add    $0x30,%rsp          ; 恢复栈指针
    1400013ea: 5d                       pop    %rbp                ; 恢复旧的栈帧基址
    1400013eb: c3                       ret                        ; 返回
```