0611-研究试验5-宣讲会研究报告-尹忠恩

> 函数如何接受不确定参数

编写`a.c`，解答 相关问题

> 1. `main` 是如何给 `showchar` 传递参数的?
> 2. `showchar` 是如何接受参数?

`a.c`

- c 代码

```c
void showchar(char a, int b);

main()
{
    showchar('a', 2);
}

void showchar(char a, int b)
{
    *(char far *)(0xb8000000 + 160 * 10 + 2 * 40) = 'a';
    *(char far *)(0xb8000000 + 160 * 10 + 2 * 40 + 1) = 2;
}
```

- 汇编代码

```
-u
076A:01FA 55            PUSH    BP
076A:01FB 8BEC          MOV     BP,SP
076A:01FD B80200        MOV     AX,0002
076A:0200 50            PUSH    AX
076A:0201 B061          MOV     AL,61
076A:0203 50            PUSH    AX
076A:0204 E80400        CALL    020B
076A:0207 59            POP     CX
076A:0208 59            POP     CX
076A:0209 5D            POP     BP
076A:020A C3            RET
```

```
076A:020B 55            PUSH    BP
076A:020C 8BEC          MOV     BP,SP
076A:020E BB00B8        MOV     BX,B800
076A:0211 8EC3          MOV     ES,BX
076A:0213 BB9006        MOV     BX,0690
076A:0216 26            ES:
076A:0217 C60761        MOV     BYTE PTR [BX],61
076A:021A BB00B8        MOV     BX,B800
076A:021D 8EC3          MOV     ES,BX
076A:021F BB9106        MOV     BX,0691
076A:0222 26            ES:
076A:0223 C60702        MOV     BYTE PTR [BX],02
076A:0226 5D            POP     BP
076A:0227 C3            RET
076A:0228 C3            RET
```

问题解答

1. `main` 函数通过 栈 给 `showchar` 传递参数

```
07C2:FFD0                               61 00 02 00 EA FF           a.....
```

2. `showchar` 应该也是从栈中获取的但是汇编代码中没有相关的栈操作。

编写 `b.c` 思考相关问题

> 1. `showchar` 函数是要如何知道呀显示多少个字符的?
> 2. `printf` 函数是如何知道有多少个参数的?

`b.c`

- c 代码

```c
void showchar(int, int, ...);

main()
{
    showchar(8, 2, 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h');
}

void showchar(int n, int color, ...)
{
    int a;
    for (a = 0; a != n; a++)
    {
        *(char far *)(0xb8000000 + 160 * 10 + 80 + a + a) = *(int *)(_BP + 8 + a + a);
        *(char far *)(0xb8000000 + 160 * 10 + 81 + a + a) = color;
    }
}
```

- 汇编代码

```
076A:01FA 55            PUSH    BP
076A:01FB 8BEC          MOV     BP,SP
076A:01FD B86800        MOV     AX,0068
076A:0200 50            PUSH    AX
076A:0201 B86700        MOV     AX,0067
076A:0204 50            PUSH    AX
076A:0205 B86600        MOV     AX,0066
076A:0208 50            PUSH    AX
076A:0209 B86500        MOV     AX,0065
076A:020C 50            PUSH    AX
076A:020D B86400        MOV     AX,0064
076A:0210 50            PUSH    AX
076A:0211 B86300        MOV     AX,0063
076A:0214 50            PUSH    AX
076A:0215 B86200        MOV     AX,0062
076A:0218 50            PUSH    AX
076A:0219 B86100        MOV     AX,0061
```

```
076A:0219 B86100        MOV     AX,0061
-u
076A:021C 50            PUSH    AX
076A:021D B80200        MOV     AX,0002
076A:0220 50            PUSH    AX
076A:0221 B80800        MOV     AX,0008
076A:0224 50            PUSH    AX
076A:0225 E80500        CALL    022D
076A:0228 83C414        ADD     SP,+14
076A:022B 5D            POP     BP
076A:022C C3            RET
076A:022D 55            PUSH    BP
076A:022E 8BEC          MOV     BP,SP
076A:0230 56            PUSH    SI
076A:0231 33F6          XOR     SI,SI
076A:0233 EB49          JMP     027E
076A:0235 8BDD          MOV     BX,BP
076A:0237 03DE          ADD     BX,SI
076A:0239 03DE          ADD     BX,SI
076A:023B 83C308        ADD     BX,+08
```

```
076A:023B 83C308        ADD     BX,+08
-u
076A:023E 8A07          MOV     AL,[BX]
076A:0240 50            PUSH    AX
076A:0241 8BC6          MOV     AX,SI
076A:0243 99            CWD
076A:0244 52            PUSH    DX
076A:0245 50            PUSH    AX
076A:0246 8BC6          MOV     AX,SI
076A:0248 99            CWD
076A:0249 5B            POP     BX
076A:024A 59            POP     CX
076A:024B 03D8          ADD     BX,AX
076A:024D 13CA          ADC     CX,DX
076A:024F 81C39006      ADD     BX,0690
076A:0253 81D100B8      ADC     CX,B800
076A:0257 8EC1          MOV     ES,CX
076A:0259 58            POP     AX
076A:025A 26            ES:
076A:025B 8807          MOV     [BX],AL
076A:025D 8A4606        MOV     AL,[BP+06]
```

```
076A:025D 8A4606        MOV     AL,[BP+06]
-u
076A:0260 50            PUSH    AX
076A:0261 8BC6          MOV     AX,SI
076A:0263 99            CWD
076A:0264 52            PUSH    DX
076A:0265 50            PUSH    AX
076A:0266 8BC6          MOV     AX,SI
076A:0268 99            CWD
076A:0269 5B            POP     BX
076A:026A 59            POP     CX
076A:026B 03D8          ADD     BX,AX
076A:026D 13CA          ADC     CX,DX
076A:026F 81C39106      ADD     BX,0691
076A:0273 81D100B8      ADC     CX,B800
076A:0277 8EC1          MOV     ES,CX
076A:0279 58            POP     AX
076A:027A 26            ES:
076A:027B 8807          MOV     [BX],AL
076A:027D 46            INC     SI
076A:027E 3B7604        CMP     SI,[BP+04]
```

```
-u
076A:0281 75B2          JNZ     0235
076A:0283 5E            POP     SI
076A:0284 5D            POP     BP
076A:0285 C3            RET
076A:0286 C3            RET
```

`printf` 函数是如何知道有多少给参数

- 先写一个 `printf.c` 的测试用例

```
1   main(){
2       printf("%c,%c,%c,%c",'a','b','c','d');
3       printf("%d,%d,%d,%d",1,2,3,4);
4   }
```

- 汇编代码
    - 第一个 `printf`

```
076A:01FA 55            PUSH    BP
076A:01FB 8BEC          MOV     BP,SP
076A:01FD B86400        MOV     AX,0064        d
076A:0200 50            PUSH    AX
076A:0201 B86300        MOV     AX,0063        c
076A:0204 50            PUSH    AX
076A:0205 B86200        MOV     AX,0062        b
076A:0208 50            PUSH    AX
076A:0209 B86100        MOV     AX,0061        a
076A:020C 50            PUSH    AX
076A:020D B89401        MOV     AX,0194        为什么要压这个?
076A:0210 50            PUSH    AX
076A:0211 E8D308        CALL    0AE7
```

    - 第二个 `printf`

```
-u 0214
076A:0214 83C40A        ADD     SP,+0A
076A:0217 B80400        MOV     AX,0004
076A:021A 50            PUSH    AX
076A:021B B80300        MOV     AX,0003
076A:021E 50            PUSH    AX
076A:021F B80200        MOV     AX,0002
076A:0222 50            PUSH    AX
076A:0223 B80100        MOV     AX,0001
076A:0226 50            PUSH    AX
076A:0227 B8A001        MOV     AX,01A0
076A:022A 50            PUSH    AX
076A:022B E8B908        CALL    0AE7
```

栈顶指针复位，前面在栈中压入了5个字
数字 4

数字 3

数字 2

数字 1

不知道?

- 通过对照 `printf` 函数可以猜想 `mov ax,0194;` 和 `mov ax 01a0` 因该是 `printf` 的第一个参数，然后通过 `debug` 可以验证猜想是正确的

```
-d ds:0194
08A3:0190                25 63 2C 25-63 2C 25 63 2C 25 63 00      %c,%c,%c,%c.
08A3:01A0  25 64 2C 25 64 2C 25 64-2C 25 64 00 00 00 00 13  %d,%d,%d,%d.....
```

通过观察可以看到 `25 63 2c 25 63 2c 25 63 2c 25 63 2c 25 63 00` 对应字符串 `%c,%c,%c,%c` 和 `25 64 2c 25 64 2c 25 64 2c 25 64 00` 对应字符串 `%d,%d,%d,%d`,, 故根据以往编程经验猜想 `00` 为终止条件。所以把第三个 `%c` 对应的汇编代码开头改变为 `00`，看看是不是只输出两个字符 通过下图可以看出猜想正确。所以可以得出 `printf` 可能是根据传入的 `%` 的个数来确定打印的字符数，读入一个 `%` 就会读取后面一个字符来确定打印的方式，当读出一个 `0` 时打印结束

```
-d ds:0194
08A3:0190                25 63 2C 25-63 2C 25 63 2C 25 63 00      %c,%c,%c,%c.
08A3:01A0  25 64 2C 25 64 2C 25 64-2C 25 64 00 00 00 00 13  %d,%d,%d,%d.....
08A3:01B0  02 02 04 05 06 08 08 08-14 15 05 13 FF 16 05 11  ................
08A3:01C0  02 FF FF FF FF FF FF FF-FF FF FF FF FF FF 05 05  ................
08A3:01D0  FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF  ................
08A3:01E0  0F FF 23 02 FF 0F FF FF-FF FF 13 FF FF 02 02 05  ..#.............
08A3:01F0  0F 02 FF FF FF 13 FF FF-FF FF FF FF FF FF 23 FF  ..............#.
08A3:0200  FF FF FF 23 FF 13 FF 00-6E 02 6E 02 6E 02 00 00  ...#....n.n.n...
08A3:0210  00 10 00 00                                      ....
-e ds:019a
08A3:019A  25.00

-p
a,b,
AX=0004  BX=FFFE  CX=0000  DX=0000  SP=FFD6  BP=FFE0  SI=003A  DI=04BD
DS=08A3  ES=08A3  SS=08A3  CS=076A  IP=0214   NV UP EI PL NZ NA PO NC
076A:0214 83C40A        ADD     SP,+0A
```

问题解答

1. 第一个参数 `n` 就是告知函数 `showchar` 显示多少给字符
2. `printf` 是通过 `%` 的个数来得知有多少个参数，通过 `00` 来判断结束。

实现一个简单的 `printf` 函数只需支持 `%c,%d` 即可

```c
void myPrintf(char *, ...);

main()
{
    myPrintf("xxxxx%c,%n,ddddd%d", 'x', 5);
}

void myPrintf(char *str, ...)
{
    int stackIndex = 0;
    int stringIndex = 0;
```

```
12      int screenIndex = 0;
13      int screenBenchmark = 160 * 10;
14
15      while (str[stringIndex] != 0)
16      {
17          if (str[stringIndex] == '%')
18          {
19              if (str[stringIndex + 1] == 'c')
20              {
21                  *(char far *)(0xb8000000 + screenBenchmark + screenIndex) = *
    (char *)(_BP + 6 + stackIndex); /*跨 push call 第一个参数 才能取到相应的值*/
22                  *(char far *)(0xb8000000 + screenBenchmark + screenIndex + 1) =
    2;
23                  screenIndex += 2;
24                  stringIndex += 2;
25                  stackIndex += 2;
26              }
27              else if (str[stringIndex + 1] == 'd')
28              {
29                  *(char far *)(0xb8000000 + screenBenchmark + screenIndex) = *
    (char *)(_BP + 6 + stackIndex) + 0x30;
30                  *(char far *)(0xb8000000 + screenBenchmark + screenIndex + 1) =
    2;
31
32                  stackIndex += 2;
33                  screenIndex += 2;
34                  stringIndex += 2;
35              }
36              else if (str[stringIndex + 1] == 'n')
37              {
38                  screenBenchmark += 160;
39                  screenIndex = 0;
40                  stringIndex += 2;
41              }
42          }
43          else
44          {
45              *(char far *)(0xb8000000 + screenBenchmark + screenIndex) =
    str[stringIndex];
46              *(char far *)(0xb8000000 + screenBenchmark + screenIndex + 1) = 2;
47
48              screenIndex += 2;
49              stringIndex += 1;
50          }
51      }
52  }
```