内存管理

实现一个简单的动态内存管理器。

预备知识

在介绍分配算法之前，需要先了解该算法的一些基本组成部分：

- 首部（header）：位于内存块的起始处，占 4 个字节。细分为两个部分：前 29 个二进制位表示对应内存块的**实际可用大小**（以 8 字节为单位。也就是说，如果这个部分的值为 3，表示内存块的实际可用大小为 3 × 8 = 24 字节）；后 3 个二进制位用于标志本内存块是否是空闲的（000 表示空闲，001 表示已分配）
- 尾部（footer）：内容与首部相同，但是位于内存块的结尾处
- 初始时，整个堆内存中除了首部和尾部的其余字节应均为 `0xdf`

也就是说，一个内存块的内容应该如下所示（一个正方形表示一个字节）：

0x00	00	00	00	08
0x04	df	df	df	df
0x08	df	df	df	df
0x0c	00	00	00	08
0x10				

如前所述，这个内存块表示一个**实际可用大小**为 **8 字节**（0x04 到 0x0b 这 8 个字节）的**空闲**内存块。

内存管理算法

内存分配

内存的分配采用最先适配（first-fit）的方式，即，对于一个分配 B 字节的请求，**从低地址向高地址**找到**第一个实际可用大小**为 S 的满足 $B \leq S$ 且空闲的内存块，如果没有满足条件的内存块，则打印错误消息。

内存释放

将对应内存块首部和尾部的**标志位**设为 0，**其他不变。**

输入描述

输入的第一行为两个整数，分别表示堆内存的大小（包括初始时首部和尾部所占的 8 字节在内，保证为正数且是 8 的整数倍）和最多分配多少个对象（对象序号为**从 1 开始**的连续整数）。

输入的第二行为一个整数 N，表示接下来有 N 条命令（保证接下来一定有 N 条命令）。

剩下的 N 行每行是一条命令，分为三种：

- 请求分配内存的 `ALLOC <object-id> <object-size>`，其中 `<object-id>` **从 1 开始**，`<object-size>` 为正整数, 单位为**字节**且**不保证**是 8 的整数倍（所以你需要一种方式将 `<object-size>` 向上取整到 8 的整数倍，否则无法使用前面提到的首部结构的方案）。保证不会在分配成功之后对同样的对象再次分配。
- 请求释放内存的 `FREE <object-id>`，其中 `<object-id>` 的含义同上。
- 请求打印整个堆内存的 `SHOW`。

保证输入的所有整数在 `int` 范围内。

输出描述

对于 `ALLOC`，如果找到了符合条件的内存块，输出 `succeeded to alloc object <object-id>`，否则输出 `failed to alloc object <object-id>`。

对于 `FREE`，如果 `<object-id>` 所标识的对象是已分配的，则输出 `succeeded to free object <object-id>`，否则输出 `invalid memory access`。

对于 `SHOW`，按照十六进制（不需要 `0x` 前缀）输出整个堆内存的内容，每行输出4个字节。

示例

示例 1

输入

```
64 5
9
ALLOC 1 1
ALLOC 2 2
ALLOC 3 3
ALLOC 4 8
FREE 1
FREE 2
FREE 3
ALLOC 5 10
SHOW
```

输出

```
succeeded to alloc object 1
succeeded to alloc object 2
succeeded to alloc object 3
succeeded to alloc object 4
succeeded to free object 1
succeeded to free object 2
succeeded to free object 3
failed to alloc object 5
00000008
dfdfdfdf
dfdfdfdf
00000008
00000008
dfdfdfdf
dfdfdfdf
00000008
00000008
dfdfdfdf
dfdfdfdf
00000008
00000009
dfdfdfdf
dfdfdfdf
00000009
```

解释

共分配了四个对象，分别占用 1、2、3 和 8 字节，由于我们要求内存块的大小必须是 8 的整数倍，所以向上取整全部得到 8。此外，由于对象 1、2 和 3 均被释放，标志位为 000，所以首部和尾部均为 `0x00000008`，而对象 4 没有被释放，标志位为 001，所以首部和尾部均为 `0x00000009`。

详细过程见附录 A。

注意

输入的命令不总是按照 `ALLOC` - `FREE` - `SHOW` 的顺序。

提示

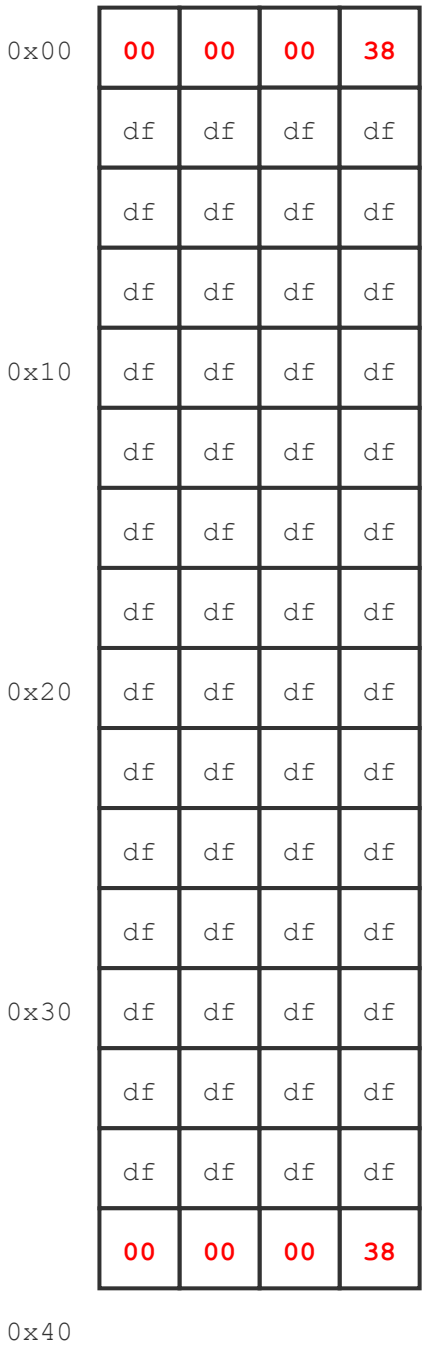
- 可以使用 `string`
- 建议先搭好基本框架，拿到部分用例的分数

附录

附录 A

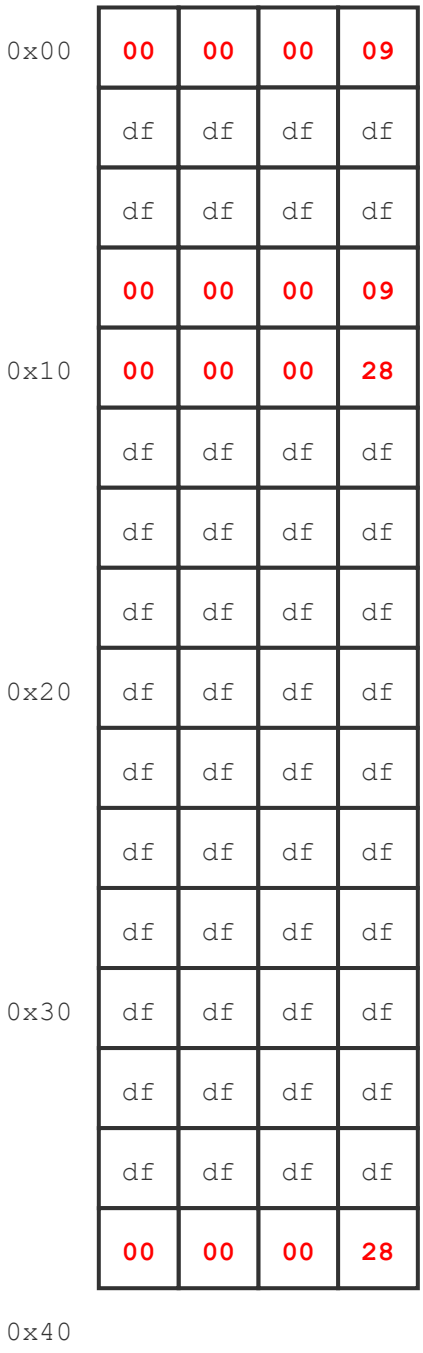
本节给出示例 1 的详细过程。

初始的堆为：



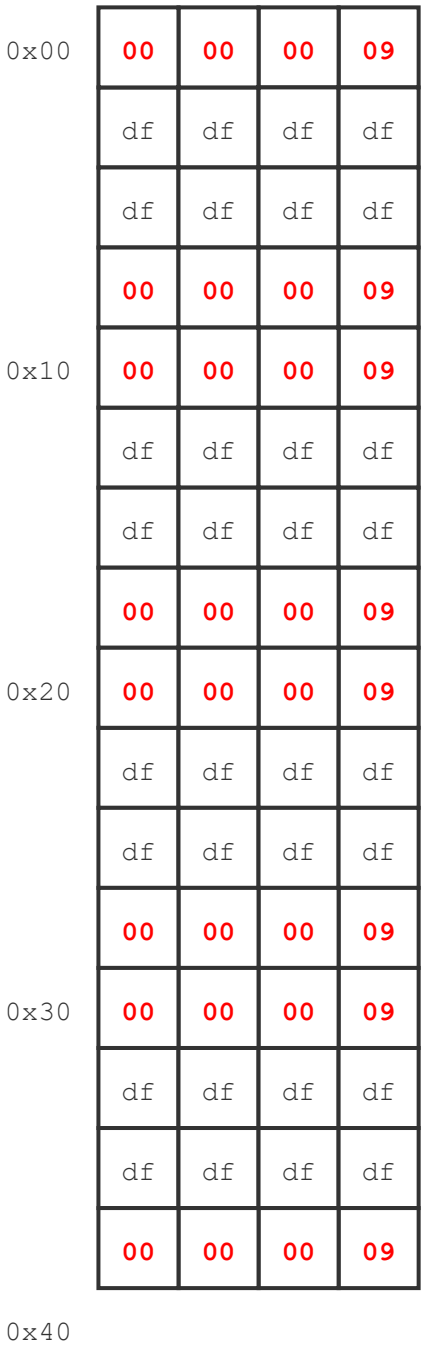
堆内存为 64 字节，其中首部和尾部各占 4 字节，这样可用内存还剩下 56 字节，用十六进制表示为 `0x38`，同时这块内存是空闲的，标志位为 0。

`ALLOC 1 1` 后：

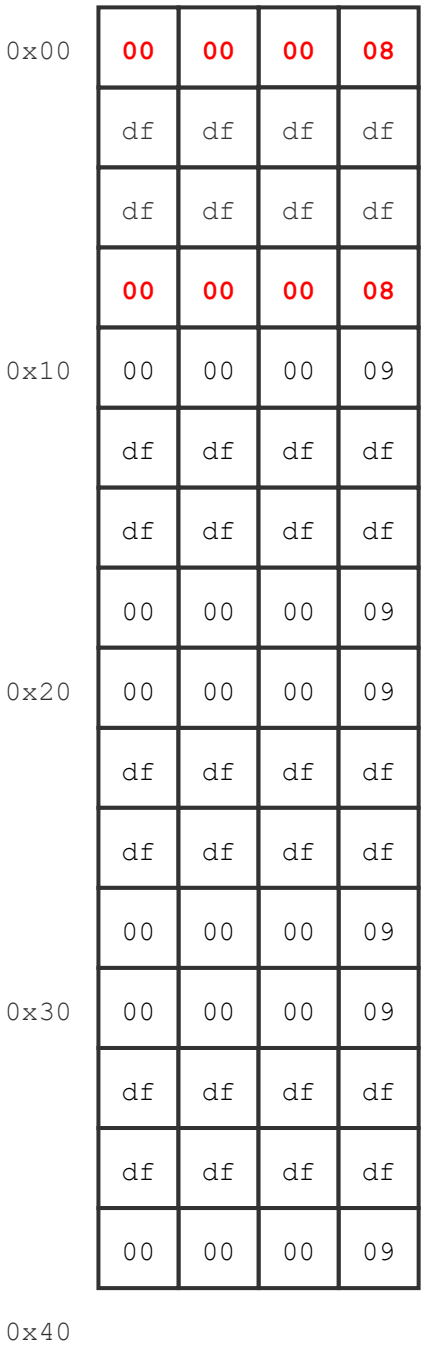


为对象 1 分配的内存块的可用内存大小为 8 字节（即，1000），最低三个二进制位为 001 表示该内存块已分配，合起来就是 `1001`，即 9。剩余的内存为 48 字节，其中首部和尾部各占 4 字节，可用内存还有 40 字节，用十六进制表示为 `0x28`，同时这块内存是空闲的，标志位为 0。

依次进行 `ALLOC 2 2`、`ALLOC 3 3` 和 `ALLOC 4 8` 后：


0x00
00 00 00 09
df df df df
df df df df
00 00 00 09
0x10
00 00 00 09
df df df df
df df df df
00 00 00 09
0x20
00 00 00 09
df df df df
df df df df
00 00 00 09
0x30
00 00 00 09
df df df df
df df df df
00 00 00 09
0x40

`FREE 1` 后:



再进行 `FREE 2` 和 `FREE 3` 就得到示例 1 的输出结果。

`ALLOC 5 10`（需要分配 16 字节）时，由于没有任何一个空闲块的实际可用大小满足条件，同时不允许合并空闲块，所以分配失败。