- 图 9-34a：程序请求一个 4 字的块。malloc 的响应是：从空闲块的前部切出一个 4 字的块，并返回一个指向这个块的第一字的指针。
- 图 9-34b：程序请求一个 5 字的块。malloc 的响应是：从空闲块的前部分配一个 6 字的块。在本例中，malloc 在块里填充了一个额外的字，是为了保持空闲块是双字边界对齐的。
- 图 9-34c：程序请求一个 6 字的块，而 malloc 就从空闲块的前部切出一个 6 字的块。
- 图 9-34d：程序释放在图 9-34b 中分配的那个 6 字的块。注意，在调用 free 返回之后，指针 p2 仍然指向被释放了的块。应用有责任在它被一个新的 malloc 调用重新初始化之前，不再使用 p2。
- 图 9-34e：程序请求一个 2 字的块。在这种情况中，malloc 分配在前一步中被释放了的块的一部分，并返回一个指向这个新块的指针。

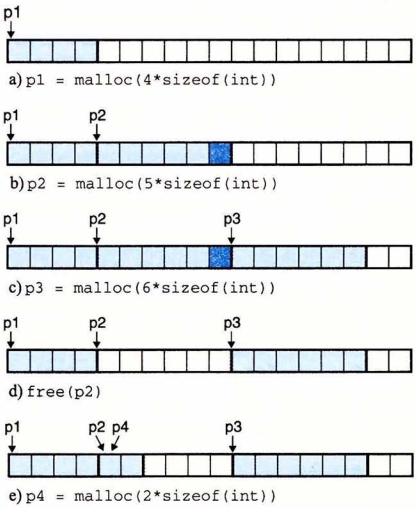

图 9-34　用 malloc 和 free 分配和释放块。每个方框对应于一个字。每个粗线标出的矩形对应于一个块。阴影部分是已分配的块。已分配的块的填充区域是深阴影的。无阴影部分是空闲块。堆地址是从左往右增加的

9.9.2　为什么要使用动态内存分配

程序使用动态内存分配的最重要的原因是经常直到程序实际运行时，才知道某些数据结构的大小。例如，假设要求我们编写一个 C 程序，它读一个 n 个 ASCII 码整数的链表，每一行一个整数，从 stdin 到一个 C 数组。输入是由整数 n 和接下来要读和存储到数组中的 n 个整数组成的。最简单的方法就是静态地定义这个数组，它的最大数组大小是硬编码的：

```
1   #include "csapp.h"
2   #define MAXN 15213
3
4   int array[MAXN];
5
6   int main()
7   {
8       int i, n;
9
10      scanf("%d", &n);
11      if (n > MAXN)
12          app_error("Input file too big");
13      for (i = 0; i < n; i++)
14          scanf("%d", &array[i]);
15      exit(0);
16  }
```