像这样用硬编码的大小来分配数组通常不是一种好想法。MAXN 的值是任意的，与机器上可用的虚拟内存的实际数量没有关系。而且，如果这个程序的使用者想读取一个比 MAXN 大的文件，唯一的办法就是用一个更大的 MAXN 值来重新编译这个程序。虽然对于这个简单的示例来说这不成问题，但是硬编码数组界限的出现对于拥有百万行代码和大量使用者的大型软件产品而言，会变成一场维护的噩梦。

一种更好的方法是在运行时，在已知了 n 的值之后，动态地分配这个数组。使用这种方法，数组大小的最大值就只由可用的虚拟内存数量来限制了。

```
1   #include "csapp.h"
2
3   int main()
4   {
5       int *array, i, n;
6
7       scanf("%d", &n);
8       array = (int *)Malloc(n * sizeof(int));
9       for (i = 0; i < n; i++)
10          scanf("%d", &array[i]);
11      free(array);
12      exit(0);
13  }
```

动态内存分配是一种有用而重要的编程技术。然而，为了正确而高效地使用分配器，程序员需要对它们是如何工作的有所了解。我们将在 9.11 节中讨论因为不正确地使用分配器所导致的一些可怕的错误。

9.9.3　分配器的要求和目标

显式分配器必须在一些相当严格的约束条件下工作：

- *处理任意请求序列*。一个应用可以有任意的分配请求和释放请求序列，只要满足约束条件：每个释放请求必须对应于一个当前已分配块，这个块是由一个以前的分配请求获得的。因此，分配器不可以假设分配和释放请求的顺序。例如，分配器不能假设所有的分配请求都有相匹配的释放请求，或者有相匹配的分配和空闲请求是嵌套的。
- *立即响应请求*。分配器必须立即响应分配请求。因此，不允许分配器为了提高性能重新排列或者缓冲请求。
- *只使用堆*。为了使分配器是可扩展的，分配器使用的任何非标量数据结构都必须保存在堆里。
- *对齐块(对齐要求)*。分配器必须对齐块，使得它们可以保存任何类型的数据对象。
- *不修改已分配的块*。分配器只能操作或者改变空闲块。特别是，一旦块被分配了，就不允许修改或者移动它了。因此，诸如压缩已分配块这样的技术是不允许使用的。

在这些限制条件下，分配器的编写者试图实现吞吐率最大化和内存使用率最大化，而这两个性能目标通常是相互冲突的。

- *目标 1：最大化吞吐率*。假定 n 个分配和释放请求的某种序列：

$$R_0, R_1, \cdots, R_k, \cdots, R_{n-1}$$