返回一个指针，指向 val 的首次出现：

```
1  int *search(int *p, int val)
2  {
3      while (*p && *p != val)
4          p += sizeof(int); /* Should be p++ */
5      return p;
6  }
```

然而，因为每次循环时，第 4 行都把指针加了 4(一个整数的字节数)，函数就不正确地扫描数组中每 4 个整数。

9.11.8　引用不存在的变量

没有太多经验的 C 程序员不理解栈的规则，有时会引用不再合法的本地变量，如下列所示：

```
1  int *stackref ()
2  {
3      int val;
4
5      return &val;
6  }
```

这个函数返回一个指针(比如说是 p)，指向栈里的一个局部变量，然后弹出它的栈帧。尽管 p 仍然指向一个合法的内存地址，但是它已经不再指向一个合法的变量了。当以后在程序中调用其他函数时，内存将重用它们的栈帧。再后来，如果程序分配某个值给 *p，那么它可能实际上正在修改另一个函数的栈帧中的一个条目，从而潜在地带来灾难性的、令人困惑的后果。

9.11.9　引用空闲堆块中的数据

一个相似的错误是引用已经被释放了的堆块中的数据。例如，考虑下面的示例，这个示例在第 6 行分配了一个整数数组 x，在第 10 行中先释放了块 x，然后在第 14 行中又引用了它：

```
1  int *heapref(int n, int m)
2  {
3      int i;
4      int *x, *y;
5
6      x = (int *)Malloc(n * sizeof(int));
7
8      . . . // Other calls to malloc and free go here
9
10     free(x);
11
12     y = (int *)Malloc(m * sizeof(int));
13     for (i = 0; i < m; i++)
14         y[i] = x[i]++; /* Oops! x[i] is a word in a free block */
15
16     return y;
17 }
```