```
typedef long data_t;
```

我们还会分配一个 len 个 data_t 类型对象的数组，来存放实际的向量元素。

图 5-4 给出的是一些生成向量、访问向量元素以及确定向量长度的基本过程。一个值得注意的重要特性是向量访问程序 get_vec_element，它会对每个向量引用进行边界检查。这段代码类似于许多其他语言(包括 Java)所使用的数组表示法。边界检查降低了程序出错的机会，但是它也会减缓程序的执行。

code/opt/vec.c

```
1  /* Create vector of specified length */
2  vec_ptr new_vec(long len)
3  {
4      /* Allocate header structure */
5      vec_ptr result = (vec_ptr) malloc(sizeof(vec_rec));
6      data_t *data = NULL;
7      if (!result)
8          return NULL;  /* Couldn't allocate storage */
9      result->len = len;
10     /* Allocate array */
11     if (len > 0) {
12         data = (data_t *)calloc(len, sizeof(data_t));
13         if (!data) {
14             free((void *) result);
15             return NULL; /* Couldn't allocate storage */
16         }
17     }
18     /* Data will either be NULL or allocated array */
19     result->data = data;
20     return result;
21 }
22
23 /*
24  * Retrieve vector element and store at dest.
25  * Return 0 (out of bounds) or 1 (successful)
26  */
27 int get_vec_element(vec_ptr v, long index, data_t *dest)
28 {
29     if (index < 0 || index >= v->len)
30         return 0;
31     *dest = v->data[index];
32     return 1;
33 }
34
35 /* Return length of vector */
36 long vec_length(vec_ptr v)
37 {
38     return v->len;
39 }
```

code/opt/vec.c

图 5-4　向量抽象数据类型的实现。在实际程序中，数据类型 data_t 被声明为 int、long、float 或 double