Now consider the following code:

double side = 3.0;
double * pd = &side;
double & rd = side;
long edge = 5L;
double lens[4] = { 2.0, 5.0, 10.0, 12.0} ;
double c1 = refcube(side);          // ra is side
double c2 = refcube(lens[2]);       // ra is lens[2]
double c3 = refcube(rd);            // ra is rd is side
double c4 = refcube(*pd);           // ra is *pd is side
double c5 = refcube(edge);          // ra is temporary variable
double c6 = refcube(7.0);           // ra is temporary variable
double c7 = refcube(side + 10.0);   // ra is temporary variable

The arguments side, lens[2], rd, and *pd are type double data objects with names, so it is possible to generate a reference for them, and no temporary variables are needed. (Recall that an element of an array behaves like a variable of the same type as the element.) But edge, although a variable, is of the wrong type. A reference to a double can't refer to a long. The arguments 7.0 and side + 10.0, on the other hand, are the right type, but are not named data objects. In each of these cases, the compiler generates a temporary, anonymous variable and makes ra refer to it. These temporary variables last for the duration of the function call, but then the compiler is free to dump them.

So why is this behavior okay for constant references but not otherwise? Recall the swapr() function of Listing 8.4:

```c
void swapr(int & a, int & b)  // use references
{
    int temp;

    temp = a;       // use a, b for values of variables
    a = b;
    b = temp;
}
```