const float g_moon = 1.63;
float * pm = &g_moon;       // INVALID

For the first case, you can use neither g_earth nor pe to change the value 9.80. C++
doesn't allow the second case for a simple reason—if you can assign the address of
g_moon to pm, then you can cheat and use pm to alter the value of g_moon. That makes
a mockery of g_moon's const status, so C++ prohibits you from assigning the address of
a const to a non-const pointer.

The situation becomes a bit more complex if you have pointers to pointers. As you saw
earlier, assigning a non-const pointer to a const pointer is okay, provided that we're
dealing with just one level of indirection:

    int age = 39;       // age++ is a valid operation
    int * pd = &age;   // *pd = 41 is a valid operation
    const int * pt = pd;  // *pt = 42 is an invalid operation

But pointer assignments that mix const and non-const in this manner no longer are safe
when you go to two levels of indirection. If it were allowed, you could do something like
this:

    const int **pp2;
    int *p1;
    const int n = 13;
    pp2 = &p1; // not allowed, but suppose it were
    *pp2 = &n; // valid, both const, but sets p1 to point at n
    *p1 = 10;  // valid, but changes const n

Here the code assigns a non-const address (&pl) to a const pointer (pp2), and that allows
pl to be used to alter const data. So the rule that you can assign a non-const address or
pointer to a const pointer only works if there is just one level of indirection, for example, if
the pointer points to a fundamental data type.

Remember

You can assign either the address of const data or
non-const data to a pointer-to-const, providing the data