Danger awaits those who incautiously use pointers. One extremely important point is that when you create a pointer in C++, the computer allocates memory to hold an address, but it does not allocate memory to hold the data to which the address points. Creating space for the data involves a separate step. Omitting that step, as in the following, is an invitation to disaster:

```c++
long * fellow;       // create a pointer-to-long
*fellow = 223323;   // place a value in never-never land
```

Sure, `fellow` is a pointer. But where does it point? The code failed to assign an address to `fellow`. So where is the value 223323 placed? We can't say. Because `fellow` wasn't initialized, it could have any value. Whatever that value is, the program interprets it as the address at which to store 223323. If `fellow` happens to have the value 1200, then the computer attempts to place the data at address 1200, even if that happens to be an address in the middle of your program code. Chances are that wherever `fellow` points, that is not where you want to put the number 223323. This kind of error can produce some of the most insidious and hard-to-trace bugs.

**Caution**

- **Pointer Golden Rule:** ALWAYS initialize a pointer to a definite and appropriate address before you apply the dereferencing operator (‘*’) to it.

**Pointers and Numbers**

Pointers are not integer types, even though computers typically handle addresses as integers. Conceptually, pointers are distinct types from integers. Integers are numbers you can add, subtract, divide, and so on. But a pointer describes a location, and it doesn't make sense, for example, to multiply two locations times each other. In terms of the operations you can perform with them, pointers and integers are different from each other. Consequently, you can't simply assign an integer to a pointer:

```c++
int * pt;
pt = 0xB8000000; // type mismatch
```

Here, the left side is a pointer to `int`, so you can assign it an address, but the right side is