points to. An address in itself reveals only the beginning address of the object stored, not its type or the number of bytes used. Look at the addresses of the two values. They are just numbers with no type or size information. Also, note that the size of a pointer-to-int is the same as the size of a pointer-to-double. Both are just addresses. But because use_new.cpp declared the pointer types, the program knows that *pd is a double value of 8 bytes, whereas *pt is an int value of 4 bytes. When use_new.cpp prints the value of *pd, cout can tell how many bytes to read and how to interpret them.

Out of Memory?

It's possible that the computer might not have sufficient memory available to satisfy a new request. When that is the case, new returns the value 0. In C++, a pointer with the value 0 is called the null pointer. C++ guarantees that the null pointer never points to valid data, so it often is used to indicate failure for operators or functions that otherwise return usable pointers. After you learn about if statements (in Chapter 6), you can check to see if new returns the null pointer and thus protects your program from attempting to exceed its bounds. In addition to returning the null pointer upon failure to allocate memory, new might throw a bad_alloc exception. Chapter 15, "Friends, Exceptions, and More," discusses the exception mechanism.

Freeing Memory with delete

Using new to request memory when you need it is just the more glamorous half of the C++ memory-management package. The other half is the delete operator, which enables you to return memory to the memory pool when you are finished with it. That is an important step toward making the most effective use of memory. Memory that you return, or free, then can be reused by other parts of your program. You use delete by following it with a pointer to a block of memory originally allocated with new:

```cpp
int * ps = new int; // allocate memory with new
... // use the memory
```