int * pn = new int;

The new int part tells the program you want some new storage suitable for holding an int.
The new operator uses the type to figure out how many bytes are needed. Then, it finds
the memory and returns the address. Next, assign the address to pn, which is declared to
be of type pointer-to-int. Now pn is the address and *pn is the value stored there.

Compare this with assigning the address of a variable to a pointer:

int higgens;
int * pt = &higgens;

In both cases (pn and pt) you assign the address of an int to a pointer. In the second case,
you also can access the int by name: higgens. In the first case, your only access is via the
pointer. That raises a question: Because the memory to which pn points lacks a name,
what do you call it? We say that pn points to a data object. This is not "object" in the sense
of "object-oriented programming"; it’s just "object" in the sense of "thing." The term "data
object" is more general than the term "variable," for it means any block of memory
allocated for a data item. Thus, a variable also is a data object, but the memory to which
pn points is not a variable. The pointer method for handling data objects may seem more
awkward at first, but it offers greater control over how your program manages memory.

The general form for obtaining and assigning memory for a single data object, which can
be a structure as well as a fundamental type, is this:

typeName pointer_name = new typeName;

You use the data type twice: once to specify the kind of memory requested and once to
declare a suitable pointer. Of course, if you've already declared a pointer of the correct
type, you can use it rather than declare a new one. Listing 4.12 illustrates using new with
two different types.

Listing 4.12 use_new.cpp

// use_new.cpp _ using the new operator
#include <iostream>
using namespace std;
int main()