```
struct Ratio
{ private:
    int num, den;
};
```

So the difference between a **`class`** and a C++ **`struct`** is really just cosmetic.

10.11 POINTERS TO OBJECTS

In many applications, it is advantageous to use pointers to objects (and `structs`). Here is a simple example:

EXAMPLE 10.12 Using Pointers to Objects

```
class X
{ public:
    int data;
};
int main()
{ X* p = new X;
  (*p).data = 22;          // equivalent to: p->data = 22;
  cout << "(*p).data = " << (*p).data << " = " << p->data << endl;
  p->data = 44;
  cout << "  p->data = " << (*p).data << " = " << p->data << endl;
}
(*p).data = 22 = 22
  p->data = 44 = 44
```

Since `p` is a pointer to an `X` object, `*p` is an `X` object, and `(*p).data` accesses its **`public`** member `data`. Note that parentheses are required in the expression `(*p).data` because the direct member selection operator "**`.`**" has higher precedence than the dereferencing operator "**`*`**". (See Appendix C.)

The two notations

```
(*p).data
p->data
```

have the same meaning. When working with pointers, the "arrow" symbol "`->`" is preferred because it is simpler and it suggests "the thing to which `p` points."

Here is a more important example:

EXAMPLE 10.13 A `Node` Class for Linked Lists

```
class Node
{ public:
    Node(int d, Node* p=0) : data(d), next(p) { }
    int data;
    Node* next;
};
```

This defines a `Node` class each of whose objects contain an `int` data member and a `next` pointer.

```
int main()
{ int n;
  Node* p;
```