that enumerated type as well as the values of its enumerators. Although implicit conversion *to* an enumerated type is never permitted, when implicitly converting *from* a classic **enum** type to some arithmetic type, the **enum** promotes to integral types in a way similar to how its underlying type would promote using the rules of **integral promotion** and **standard conversion**:

```
void f()
{
    enum A { e_A0, e_A1, e_A2 };  // classic, C-style C++03 enum
    enum B { e_B0, e_B1, e_B2 };  //    "         "     "     "

    A a;  // Declare object a to be of type A.
    B b;  //    "       "   b  "  "  "   "  B.

    a = e_B2;  // Error, cannot convert e_B2 to enum type A
    b = e_B2;  // OK, assign the value e_B2 (numerically 2) to b.
    a = b;     // Error, cannot convert enum type B to enum type A
    b = b;     // OK, self-assignment
    a = 1;     // Error, invalid conversion from int 1 to enum type A
    a = 0;     // Error, invalid conversion from int 0 to enum type A

    bool     v = a;     // OK
    char     w = e_A0;  // OK
    int      i = e_B0;  // OK
    unsigned y = e_B1;  // OK
    float    x = b;     // OK
    double   z = e_A2;  // OK
    char*    p = e_B0;  // Error, unable to convert e_B0 to char*
    char*    q = +e_B0; // Error, invalid conversion of int to char*
}
```

Notice that, in this example, the final two diagnostics for the attempted initializations of `p` and `q`, respectively, differ slightly. In the first, we are trying to initialize a pointer, `p`, with an enumerated type, `B`. In the second, we have creatively used the built-in unary-plus operator to explicitly promote the enumerator to an integral type before attempting to assign it to a pointer, `q`. Even though the numerical value of the enumerator is `0` and such is known at compile time, implicit conversion to a pointer type from anything but the literal integer constant `0` is not permitted. Excluding esoteric user-defined types, only a literal `0` or, as of C++11, a value of type `std::nullptr_t` is implicitly convertible to an arbitrary pointer type; see Section 1.1."**nullptr**" on page 99.

C++ fully supports comparing values of *classic* **enum** types with values of arbitrary **arithmetic type** as well as those of the same enumerated type; the operands of a comparator will be promoted to a sufficiently large ~~integer~~ type, and the comparison will be done with