**Enumerated Types:**

enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES};

The above C++ statement declared a user-defined type called an enumeration. It starts with the keyword **enum** followed by a type name, *Suit,* and a list of enumeration constants. The values of these constants are integers starting at 0, unless specified otherwise, and increment by 1 throughout the list. You can use the enumerations constants directly in your C++ code as in the following example.

Suit suit = SPADES;

If (suit == CLUBS) {…}

If you want the enumeration to start at a value different from 0, then explicitly set the value in the list as in the following.

enum Rank {TWO = 2, THREE, FOUR, …, JACK, QUEEN, KING, ACE };

**Iterating through an enumerated type:** Ideally you would like to iterate through an enumerated type as follows.

for (Rank r = TWO; r <= ACE; r++)

The problem is that for some strange reason, the increment operator ++ is not defined for enumerated types! We could fix this by *overloading* the ++ operator so that it does work for our enumerated type but *operator overloading* is the topic of chapter 11. Instead we can use the less aesthetic method of adding 1 and casting it back to the enumerated type.

for (Rank r = TWO; r <= ACE; r = Rank(r+1) )

The expression r+1 adds one to the integer constant r, then Rank() casts its type back to Rank. A problem occurs when trying to increment an enumerated type past its last defined value, e.g. when r is ACE in the above loop and we try to increment it to the next Rank. We can fix this by adding another Rank, say NO\_RANK, to our enum. Then there is a value past ACE that we can increment to.