Number has two members, kind and u. The value of kind will be either INT_KIND or DOUBLE_KIND.

Each time we assign a value to a member of u, we'll also change kind to remind us which member of u we modified. For example, if n is a Number variable, an assignment to the i member of u would have the following appearance:

```
n.kind = INT_KIND;
n.u.i = 82;
```

Notice that assigning to i requires that we first select the u member of n, then the i member of u.

When we need to retrieve the number stored in a Number variable, kind will tell us which member of the union was the last to be assigned a value. The print_number function can take advantage of this capability:

```
void print_number(Number n)
{
  if (n.kind == INT_KIND)
    printf("%d", n.u.i);
  else
    printf("%g", n.u.d);
}
```

⚠ It's the program's responsibility to change the tag field each time an assignment is made to a member of the union.

16.5 Enumerations

In many programs, we'll need variables that have only a small set of meaningful values. A Boolean variable, for example, should have only two possible values: "true" and "false." A variable that stores the suit of a playing card should have only four potential values: "clubs," "diamonds," "hearts," and "spades." The obvious way to deal with such a variable is to declare it as an integer and have a set of codes that represent the possible values of the variable:

```
int s;    /* s will store a suit */
...
s = 2;    /* 2 represents "hearts" */
```

Although this technique works, it leaves much to be desired. Someone reading the program can't tell that s has only four possible values, and the significance of 2 isn't immediately apparent.

Using macros to define a suit "type" and names for the various suits is a step in the right direction: