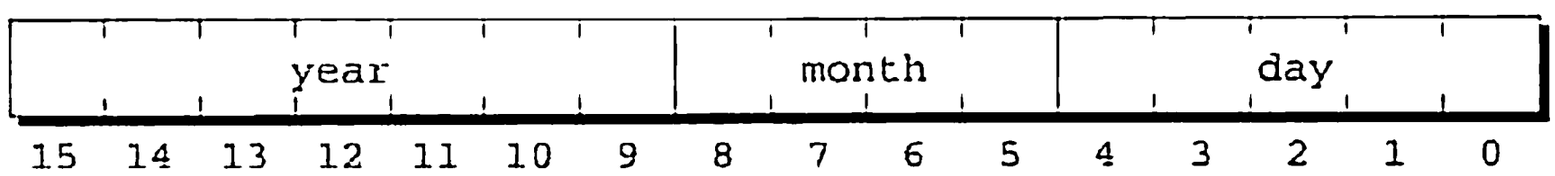
20.2 Bit-Fields in Structures

Although the techniques of Section 20.1 allow us to work with bit-fields, these techniques can be tricky to use and potentially confusing. Fortunately, C provides an alternative: declaring structures whose members represent bit-fields.

As an example, let's look at how the MS-DOS operating system (often just called DOS) stores the date at which a file was created or last modified. Since days, months, and years are small numbers, storing them as normal integers would waste space. Instead, DOS allocates only 16 bits for a date, with 5 bits for the day, 4 bits for the month, and 7 bits for the year:



Using bit-fields, we can define a C structure with an identical layout:

```
struct file_date {
  unsigned int day: 5;
  unsigned int month: 4;
  unsigned int year: 7;
};
```

The number after each member indicates its length in bits. Since the members all have the same type, we can condense the declaration if we want:

```
struct file_date {
  unsigned int day: 5, month: 4, year: 7;
};
```

The type of a bit-field must be either `int`, `unsigned int`, or `signed int`. Using `int` is ambiguous; some compilers treat the field's high-order bit as a sign bit, but others don't.

portability tip *Declare all bit-fields to be either `unsigned int` or `signed int`.*

C99 In C99, bit-fields may also have type `_Bool`. C99 compilers may allow additional bit-field types.

We can use a bit-field just like any other member of a structure, as the following example shows:

```
struct file_date fd;

fd.day = 28;
fd.month = 12;
fd.year = 8;      /* represents 1988 */
```

Note that the `year` member is stored relative to 1980 (the year the world began.