Q & A

Q: You said that C wasn't designed for writing large programs. Isn't UNIX a large program? [p. 483]

A: Not at the time C was designed. In a 1978 paper, Ken Thompson estimated that the UNIX kernel was about 10,000 lines of C code (plus a small amount of assembler). Other components of UNIX were of comparable size; in another 1978 paper, Dennis Ritchie and colleagues put the size of the PDP-11 C compiler at 9660 lines. By today's standards, these are indeed small programs.

Q: Are there any abstract data types in the C library?

A: Technically there aren't, but a few come close, including the `FILE` type (defined in `<stdio.h>`). Before performing an operation on a file, we must declare a variable of type `FILE *`:

FILE type ➤22.1

```
FILE *fp;
```

The `fp` variable will then be passed to various file-handling functions.

Programmers are expected to treat `FILE` as an abstraction. It's not necessary to know what a `FILE` is in order to use the `FILE` type. Presumably `FILE` is a structure type, but the C standard doesn't even guarantee that. In fact, it's better not to know too much about how `FILE` values are stored, since the definition of the `FILE` type can (and often does) vary from one C compiler to another.

Of course, we can always look in the `stdio.h` file and see what a `FILE` is. Having done so, there's nothing to prevent us from writing code to access the internals of a `FILE`. For example, we might discover that `FILE` is a structure with a member named `bsize` (the file's buffer size):

```
typedef struct {
  ...
  int bsize;    /* buffer size */
  ...
} FILE;
```

Once we know about the `bsize` member, there's nothing to prevent us from accessing the buffer size for a particular file:

```
printf("Buffer size: %d\n", fp->bsize);
```

Doing so isn't a good idea, however, because other C compilers might store the buffer size under a different name, or keep track of it in some entirely different way. Changing the `bsize` member is an even worse idea:

```
fp->bsize = 1024;
```

Unless we know all the details about how files are stored, this is a dangerous thing to do. Even if we *do* know the details, they may change with a different compiler or the next release of the same compiler.