- Not that it's worth mentioning, but the endless `while` loop setup, equivalent to `for(;;)`, is written `while(1)`. In either case, the statements belonging to the loop are repeated indefinitely or until a `break` statement frees things up.

C from the inside out

Although C can be a strict language, it can also be flexible. For example, just because a function returns a value doesn't mean that you have to store that value in a variable. You can use the value immediately inside another function.

As an example, consider the `getchar()` function, which returns a character typed at the keyboard. You can use that character immediately and not store it in a variable. That's what I call "using C inside out." It's one of the more flexible things you can do with C.

The TYPER2.C program is a useful one to illustrate the example of C code being written from the inside out. Here's what I mean:

```
while(ch!='~')
      ch=getchar();
```

The variable *ch* is set by the `getchar()` function. Or, put another way, the `getchar()` function generates the key press. The `ch` is just a holding place, and using `ch` to store that value is merely an intermediate step. To wit:

```
while(getchar()!='~')
      ;
```

The drawback is that the character generated by `getchar()` isn't saved anywhere. But, it does illustrate a nifty aspect of both the C language and the `while` loop.

Reedit the TYPER2.C source code so that it looks like this:

```
#include <stdio.h>

int main()
{
    puts("Start typing.");
    puts("Press ~ then Enter to stop");

    while(getchar() != '~')
        ;
    printf("Thanks!\n");
    return(0);
}
```