```
orig_handler = signal(SIGINT, handler);
```

This statement installs `handler` as the handler for `SIGINT` and then saves a pointer to the original handler in the `orig_handler` variable. To restore the original handler later, we'd write

```
signal(SIGINT, orig_handler); /* restores original handler */
```

Predefined Signal Handlers

Instead of writing our own signal handlers, we have the option of using one of the predefined handlers that `<signal.h>` provides. There are two of these, each represented by a macro:

- **`SIG_DFL`.** `SIG_DFL` handles signals in a "default" way. To install `SIG_DFL`, we'd use a call such as

  ```
  signal(SIGINT, SIG_DFL);    /* use default handler */
  ```

  The effect of calling `SIG_DFL` is implementation-defined, but in most cases it causes program termination.

- **`SIG_IGN`.** The call

  ```
  signal(SIGINT, SIG_IGN);    /* ignore SIGINT signal */
  ```

  specifies that `SIGINT` is to be ignored if it should be raised later.

In addition to `SIG_DFL` and `SIG_IGN`, the `<signal.h>` header may provide other signal handlers; their names must begin with `SIG_` followed by an uppercase letter. At the beginning of program execution, the handler for each signal is initialized to either `SIG_DFL` or `SIG_IGN`, depending on the implementation.

`<signal.h>` defines another macro, `SIG_ERR`, that looks like it should be a signal handler. `SIG_ERR` is actually used to test for an error when installing a signal handler. If a call of `signal` is unsuccessful—it can't install a handler for the specified signal—it returns `SIG_ERR` and stores a positive value in `errno`. Thus, to test whether `signal` has failed, we could write

```
if (signal(SIGINT, handler) == SIG_ERR) {
  perror("signal(SIGINT, handler) failed");
  ...
}
```

There's one tricky aspect to the entire signal-handling mechanism: what happens if a signal is raised by the function that handles that signal? To prevent infinite recursion, the C89 standard prescribes a two-step process when a signal is raised for which a signal-handling function has been installed by the programmer. First, either the handler for that signal is reset to `SIG_DFL` (the default handler) or else the signal is blocked from occurring while the handler is executing. (`SIGILL` is a special case; neither action is required when `SIGILL` is raised.) Only then is the handler provided by the programmer called.