```
    else {
      printf("Program terminates: longjmp called\n");
      return 0;
    }

    f1();
    printf("Program terminates normally\n");
    return 0;
}

void f1(void)
{
    printf("f1 begins\n");
    f2();
    printf("f1 returns\n");
}

void f2(void)
{
    printf("f2 begins\n");
    longjmp(env, 1);
    printf("f2 returns\n");
}
```

The output of this program will be

```
setjmp returned 0
f1 begins
f2 begins
Program terminates: longjmp called
```

The original call of `setjmp` returns 0, so `main` calls `f1`. Next, `f1` calls `f2`, which uses `longjmp` to transfer control back to `main` instead of returning to `f1`. When `longjmp` is executed, control goes back to the `setjmp` call. This time, `setjmp` returns 1 (the value specified in the `longjmp` call).

Q & A

Q: **You said that it's important to store zero in `errno` before calling a library function that may change it, but I've seen UNIX programs that test `errno` without ever setting it to zero. What's the story? [p. 629]**

A: UNIX programs often contain calls of functions that belong to the operating system. These ***system calls*** rely on `errno`, but they use it in a slightly different way than described in this chapter. When such a call fails, it returns a special value (such as –1 or a null pointer) in addition to storing a value in `errno`. Programs don't need to store zero in `errno` before such a call, because the function's return value alone indicates that an error occurred. Some functions in the C standard library work this way as well, using `errno` not so much to signal an error as to specify which error it was.