Exercises

7.1 On an Intel x86 system under Linux, if we execute the program that prints "hello, world" and do not call `exit` or `return`, the termination status of the program—which we can examine with the shell—is 13. Why?

7.2 When is the output from the `printfs` in Figure 7.3 actually output?

7.3 Is there any way for a function that is called by `main` to examine the command-line arguments without (a) passing `argc` and `argv` as arguments from `main` to the function or (b) having `main` copy `argc` and `argv` into global variables?

7.4 Some UNIX system implementations purposely arrange that, when a program is executed, location 0 in the data segment is not accessible. Why?

7.5 Use the `typedef` facility of C to define a new data type `Exitfunc` for an exit handler. Redo the prototype for `atexit` using this data type.

7.6 If we allocate an array of `long`s using `calloc`, is the array initialized to 0? If we allocate an array of pointers using `calloc`, is the array initialized to null pointers?

7.7 In the output from the `size` command at the end of Section 7.6, why aren't any sizes given for the heap and the stack?

7.8 In Section 7.7, the two file sizes (879443 and 8378) don't equal the sums of their respective text and data sizes. Why?

7.9 In Section 7.7, why does the size of the executable file differ so dramatically when we use shared libraries for such a trivial program?

7.10 At the end of Section 7.10, we showed how a function can't return a pointer to an automatic variable. Is the following code correct?

```
int
f1(int val)
{
    int     num = 0;
    int     *ptr = &num;

    if (val == 0) {
        int     val;

        val = 5;
        ptr = &val;
    }
    return(*ptr + 1);
}
```